**[Type the company name]**

13

**LinuxCBT-Bash**

**Dragan**

Kazalo

[OSNOVE: 4](#_Toc384073341)

[Linux uporabne osnove 4](#_Toc384073342)

[DODAJANJE NOVEGA UPORABNIKA 5](#_Toc384073343)

[ODSTRANJEVANJE UPORABNIKA 5](#_Toc384073344)

[INFORMACIJE O UPORABNIKU (UID, GID...) 5](#_Toc384073345)

[UREJANJE UPORABNIKA 5](#_Toc384073346)

[USER ID (UID) DEFINICIJA 5](#_Toc384073347)

[TESTI 6](#_Toc384073348)

[BASH KOMANDE: 7](#_Toc384073349)

[BUILT IN COMMANDS: 11](#_Toc384073350)

[SHELL EXPANSIONS - Brace expansion 11](#_Toc384073351)

[ALIASES 11](#_Toc384073352)

[EXIT STATUS 12](#_Toc384073353)

[COMMAND CHAINING AND COMMAND LISTS 12](#_Toc384073354)

[I\O REDIRECTION 12](#_Toc384073355)

[PIPES 13](#_Toc384073356)

[COMMAND SUBSTITUTIONS 13](#_Toc384073357)

[QUOTING NUANCES 13](#_Toc384073358)

[HELLO WORLD 14](#_Toc384073359)

[FUNCTIONS: 15](#_Toc384073360)

[FOOR LOOPS: 16](#_Toc384073361)

[WHILE LOOPS 17](#_Toc384073362)

[UNTILL LOOP 18](#_Toc384073363)

[CONTROL STRUCTURES 19](#_Toc384073364)

[CONTROL STRUCTURES 2: 20](#_Toc384073365)

[CONTROL STRUCTURES 3 23](#_Toc384073366)

[POSITIONAL PARAMETERS 24](#_Toc384073367)

[SELECT MENUS 25](#_Toc384073368)

[MOVE MANY FILES 26](#_Toc384073369)

[NETWORK CHECK 27](#_Toc384073370)

[FILE DIFFERENCES 28](#_Toc384073371)

[MONITOR SERVICE 30](#_Toc384073372)

[FTP SYNCHRONISATION 31](#_Toc384073373)

[PARSE LOGS 32](#_Toc384073374)

[BACKUP FILES 1 33](#_Toc384073375)

[LOGGING TECHNIQUES 35](#_Toc384073376)

[LOG ABSTRACT 36](#_Toc384073377)

### OSNOVE:

#### Linux uporabne osnove

V terminalu vidiš ali je logiran root ali pa običen uporabnik po tem, da ima root desno od imena naprave "#", user pa $.

V /etc/profile je glavni bash config fajl. Pregledaš z "less /etc/profile". Tu notri daješ globalne spremenljivke (male, logname, hostname).

Kritične spremenljivke navadno označimo z velikimi začetnicami (npr. HISTSIZE=1000).

Drugi pomemben fajl je /etc/bashrc. Ta vsebuje funkcije in aliase. Kakršnokoli logiko, ki bi jo rad izvajal na uporabniku, ki je trenutno prijavljen, delaš tu notri (funkcije, logike, aliasi).

Alias: dodeliš alias določeni komandi (npr komandi ls -la dodeliš alias ls), tako da ti ni treba pisati celotne komande.

V UNIX okolju, so fajli s prefixom "." (pika), hidden fajli. Da hidden fajle vidiš, vpiši komando ls -al.

/etc/skel ima skrite fajle. .bash\_logout pač pove, kaj se zgodi, ko se iz sistema odjavimo. .bash\_profile overrida nastavitve za specifičnega userja, ki so nastavljene v globalnem /etc/profile. Kakršne koli spremembe narediš v /etc/skel se podedujejo šele z novimi uporabniki. Te konfiguracijski fajli iz /etc/skel se avtomatsko skopirajo v /home mapo novega userja. Trenutni uporabniki ne dobijo nobenih sprememb. Da bi vplivali na trenutne uporabnike, moramo urediti .bash\_ (v vodiču npr .bash\_profile) fajle od trenutnih uporabnikov (v RPi in kali je profile brez .bash). Da bi uredili vse uporabnike, pač uredimo /etc/profile ali pa /etc/bashrc na globalni ravni.

#### DODAJANJE NOVEGA UPORABNIKA

useradd "username" (da kreira mapo v home daš še switch -m). V kali linux oz. verjetno v novih izdajah basha se uporablja adduser, ki je uporabniku bolj prijazen.

#### ODSTRANJEVANJE UPORABNIKA

userdel "username" -r (-r izbriše še vsebino /home mape in mail).

#### INFORMACIJE O UPORABNIKU (UID, GID...)

id "username"

grep "username" /etc/passwd: (vidimo UID, GID, default home dir, !default interpreter!)

#### UREJANJE UPORABNIKA

usermod (-u spremeni UID, -c spremeni full name, ki mora biti v narekovajih, -g spremeni GID...)

#### USER ID (UID) DEFINICIJA

Je unikaten positiven integer dodeljen vsakemu userju. UIDji si shranjeni v /etc/passwd (GIDi so shranjeni v /etc/groups). Tu vidimo tudi group ID, ki je po defaultu enak UID za navadne userje.

UID 0 je vedno root. Temu accountu lahko spremenimo username, ali celo drugemu userju damo ID 0, a je to varnostno sporno.

UID 65535 je navadno rezerviran za "nobody". To je user prez sistemskih pravic. Ta ID je za posameznike, ki do sistema dostopajo oddaljeno preko FTP ali HTTP.

UID 1-99 za sistemske uporabnike (wheel, daemon, lp...). To so administratorji, ki ne rabijo root moči, a vseeno opravljajo neka administrativna dela, zato rabijo več pravic kot običajen user.

Priročno je za uporabnike rezervirati blok UIDjev npr. 1000-9999 za lokalne userje in npr. 10000 - 65534 za remote userje (userje nekje v omrežju). To olajša branje sistemskih logov za kakršnekoli sumljive aktivnosti userjev.

UIDji se za nepriviligirane userje razlikujejo glede na distribucijo linuxa.

UID ni nujno da je unikaten, ni pa to varno. Prav tako ni varno reciklirati starega UIDja.

### TESTI

Bashu poveš naj izvede test z "[" (levi oglati oklepaj). To opaziš v skriptah, ki so že po defaultu v sistemu.

Z vsako izvedeno komando v bashu, se returna exit status. Shrani se v sistemski spremenljivki $?. Z izvedbo "echo $?" preverimo ali se je komanda uspešno izvedla. V UNIXu 0 pomeni uspešno in 1 neuspešno (obratno od boolean).

***Primeri:***

Za integerje: -eq (equals to), -lt (less then), -gt (greater then), -ne (not equal to), le (less or equal to), -ge (greater or equal to), -ne (not equal to)

Za stringe: = (je enako), != (ni enako)

Za fajle: -nt (newer then), -ot (older then), -e (exists), -f (is a regular file), -S (is a socket file), -s (preveri če je fajl večji od 0B)

za folderje: -e (exists)

test 1 -eq 1 (vprašamo ali je 1 enako ena. Ni tako kot programiranje, kjer daš 1 == 1). Ko potem damo "echo $?" je rezultat 0, torej uspešno izveden test.

Test in echo komande lahko združimo skupaj (chain command, ločene komande z ";").

***Primer:***

test 2 -lt 1; echo $?

test hello\_world = hello\_world; echo $? (tu testiramo stringe, če so enaki. Pozor, za stringe se uporablja enojni =, za razliko od PHP, perl...)

test hello\_Worlds != hello\_world; echo $? (rezultat 0, stringa sta res različna)

test -b imefajla1; echo $? (-b testira ali je fajl "block device". Naprava, ki posreduje podatke po znakih. Printer, Terminal...)

test -c imefajla1: echo $? (-c testira ali je fajl character device. Naprava, ki posreduje podatke v blokih skozi buffer management kodo. Recimo da hočeš podatke s skripto posredovati skozi določen port na računaniku, najprej preveriš če je character device).

test -e imefajla3 (ali imefolderja3); echo $?

test -f imefajla3; echo $? (preveriš če je res fajl)

test -S /var/lib/mysql/mysql.sock ; echo $? (preveri, če je to socket file. Socket file je posebna vrsta fajla. Vrste fajlov vidiš s komando ls -l. Prvi znak nam pove vrsto elementa. (- normalen fajl, d mapa, s socket)

test -s imefajla3; echo $? (preveri, da fajl ni prazen, torej da je večji od 0 bytov)

*V skripti ne napišeš dejansko "test" ampak uporabiš "[".(npr.[ -f imefajla3 ] in če ta pogoj drži, se skripta izvrši). "Test" se uporablja v terminalu.*

### BASH KOMANDE:

cat; izpiše vsebino fajla (fajlov) in nato vsebino tega zapiše v en fajl. (-A pokaže vse znake, -s pokaže max 1 prazno vrstico naenkrat).

tac: obratno od cat. Izpiše vsebino fajla (lahko veših fajlov) v obratni smeri.

***Primer:***

cat file1 file2 > file3 (vsebino file1 in file2 spravi v file3)

touch: ustvari nov prazen fajl, ali pa spremeni timestamp obstoječega fajla.

ls: (-ali izpiše čisto vse kar je v direktoriju in vse informacije)

wc: prešteje vrstice, besede in characterje v fajlu (ali samo eno izmed teh npr. wc -l data1 za štetje vrstic)

***Primer:***

linecount='wc -l filename1' (prešteje vrstice filename1 in jih da v ustvarjeno spremenljivko linecount)

echo $linecount

ls -A | wc -l: (ls -A izpiše vse in samo fajle v folderju. To potem pošljemo word counterju wc da prešteje samo vrstice, in tako dobimo število fajlov v folderju)

ls --time=atime -ltr filename1 (pove nam čas zadnjega accessa tega fajla)

ls --time=ctime -ltr filename1 (pove nam kdaj je bil ta fajl spremenjen, modificiran)

Sleep 5: (skripta se preneha izvajati za 5 sekund. Vrednost je seveda poljubna. To pač postaviš nekam v skripto)

usleep 2 (skripta se preneha izvajati za 2 mikrosekundi)

time imekomande: (pove koliko časa se je komanda izvajala)

ps -aux (zlista vse procese. Največkrat uporabljamo z -aux)

***Primer:***

ps -aux | grep bash (pokaže samo procese z imenom bash)

top: (lepo prikaže vse procese, memory.... Ne uporablja se pogosto)

watch tail /var/log/messages: (v posebnem pogledu prikaže spodnji del fajla messages in zadevo konstantno refresha. Interval nastavimo z -n. Tail je razlog da gledamo spodnji del fajla. Ampak to prav pride, ker loge, ki jih želimo ponavadi pogledati, so novi, zato v spodnjem delu fajla)

date: (omogoča exportanje datuma na mnogo načinov. Prikaže kdaj je bil fajl modificiran... Date omogoča več oblik prikaza in to s switchi označenimi z %. Pred njih vedno postavimo znak +)

***Primer:***

date -r imefajla +%s: (-r pove da želimo zvedeti datum, kdaj je bil imefajla modificiran. %s pa nam pove koliko časa je preteklo od Unix epoch tima do modifikacije fajla. Unix epoch time je 1.1.1970)

touch `date +%F`.log: (ustravi fajl z imenom iz datuma in obliko %F. Na koncu pripiše še .log. Date mora biti v `` znakih altgr + 7)

echo "Program Succeded" >> `date +%F`.log: (ustvari fajl s prefixom današnji datum, vanj vpiše program succeded in fajlu pripne končnico .log)

date +%N: (random string)

Expr: izvaja operacije

***Primer:***

expr 10 + 100 (da rezultat 110. Pri tem pa=i da so med ciframi in plusom presledki)

expr 10 \\* 100 (znak za množenje ne moreš napisat samo z "\*", ker je to escape character in se v linuxu uporablja kot wildcard. Zato pred \* napišemo \)

expr 100 / 10 (deljenje)

i=1

i=`expr $i + 1` (najprej spremenljivki i damo vrednost i in nato i povečamo za 1. Expr v obratnihnarekovajih)

expr nekstring = nekstring (Primerjamo 2 stringa. Rezultat je bodisi 1 kot true ali 0 kot false. Pozor, ravno obratno kot je bilo prej pri "test")

expr length nekstring (pokaže dolžino stringa)

expr substr nekstring 1 4 (izpiše samo izbran del stringa. V tem primeru začne pri prvi črki (1). Vse skupaj mora biti 4 znakov (4).

seq: (dober način za ustvarjanje loopov v bashu. Uporablja se tudi za kreiranje velikih testnih/dummy fajlov za testiranje jobov.)

fg: prikaže background jobe

jobs: prikaže vse jobe

***Primer:***

seq 50: v vsako vrstico zapiše cifro od 1 do 50

seq 0 9: v vsako vrstico zapiše cifre od 0 - 9

seq 1000000 > testfile: s tem ustvariš job. Job pa zato, ker se izvaja dlje časa. Izvajanje joba lahko damo v background s komando ctrl + z. Če želimo job dati nazaj v foreground, vpišemo "fg". S komando "jobs" lahko pogledamo kateri jobi so v backgroundu. Job lahko ubijemo v foregroundu s ctrl + c)

seq 100000 > testfile&: Z znakom & pošljemo job direktno v background, ampak se tam še vedno izvaja. Pri tem lahko npr. damo "watch testfile" in gledamo kako se stvari zapisujejo v fajl.

sort: uredi vsebino fajla po vrstnem redu, tako da so najprej cifre po velikosti, nato črke po abecedi

***Primer:***

sort file1 | uniq: uniq je tu zato, da v primeru podvojenih vnosov odstrani duplikate

cut: iz fajla nam vzame določen del. Stvari v fajlu so ločeni z delimiterji, bodisi space, comma, semicolon...)

***Primer:***

cat /etc/passwd | cut -d: -f1,3 (-d nam pove kateri delimiter je v uporabi, torej :. -f nam pove katere fielde želimo izpisati. Field 1 je v passwd username in field 3 so UIDji. Tako dobimo spisek usernamov in njihovih UIDjev iz passwd)

tr: (character translation command. Operiramo s characterji, največkrat obdelava file nameov)

***Primeri:***

echo FILE1 | tr A-Z a-z (Z echo smo poslali string FILE1 preko pipa v obdelavo. Vse uppercase znake A-Z spremeni v lowercase a-z. Se pravi prvi A-Z je tam zato, da mu povemo, kaj naj obdela, drugi a-z pa v kaj naj obdela)

echo FILEE1 | tr -s A-Z | tr A-Z a-z (z echo pošljemo string FILEE1 v obdelavo. -s pove da želimo izbrisati podvojene characterje, zato izbriše "E". Nato ta rezultat še enkrat pošljemo v obdelavo, da dobimo lowercase znake)

echo FILEE1 | tr -s A-Z | tr [:upper:] [:lower:] (enako kot zgornji primer, le da tu uporabljamo POSIX standard obdelave uppercase/lowercase)

echo FiLE1 | tf -c A-Z a-z: (Tu pa mu povemo s switchem -c da naj NE obdela uppercasov v lowercase, vse ostalo pa lahko. Ta stavek nima veliko smisla ampak zapomni si da -s pove naj izbran izbor ignorira)

***Primer v skripti s for loopom:***

﻿﻿#1/bin/bash

#Author: Dragan

#Date: 8.8.2013

#Purpose: Prikaz tr v skripti za prtetvajanje lower v uppercase

for i in `ls -A` #(deklariramo spremenljivko i, ki bo nosila vsak element spiska.

#For zanka v bashu gre čez vsak element spiska posebej)

do

newname=`echo $i | tr A-Z a-z` #(nova spremenljivka newname, naj ima vrednost

#i, a lowercase)

mv $i $newname #(preimenuj vrednosti $i v vrednosti $newname)

done

#END

basename $0: je ime skripte ki se izvaja. $0 pa je celotna pot skripte, zato basename, da pot odstrani

***Primer:***

#1/bin/bash

#1/bin/bash

#Author: Dragan

#Date: 8.8.2013

#Purpose: Prikaz tr v skripti za prtetvajanje lower v uppercase

myscriptname=`basename $0` # Spremenljivka $myscriptname bo samo ime skripte.

for i in `ls -A` # Deklariramo spremenljivko, ki bo nosila vsak element spiska.

# For zanka v bashu, gre čez vsak element spiska posebej)

do

if [ $i = $myscriptname ] # Če je element v $i enak kot ime skripte,

# napiši text spodaj)

then

echo "Sorry, can't rename myself"

else

newname=`echo $i | tr A-Z a-z`

mv $i $newname

fi

done

#END

### BUILT IN COMMANDS:

pwd: Print working directory pove kje se nahajamo v Linux file systemu. To je sistemska spremenljivka. Da to dokažemo lahko uporabimo set | grep PWD. Ta komanda razložena malo nižje. Ko to izvedemo, nam sistem pokaže sistemsko spremenljivko PWD in zraven nje še OLDPWD. Ti dve spremenljivki sistem konstantno updejta.

cd: Pošlje nas v home mapo trenutnega userja. (cd / nas pošlje v root mapo, cd ~ nas pošlje v home mapo trenutnega userja, cd ../../ nas pošlje 2 levela gor)

read x: dodeljevanje spremenljivke x. Po vpisu read nas sistem čaka, da mu damo vrednost spremenljivke. X je lahko karkoli. Lahko tudi več spremenljivk naenkrat. Njihova imena in vrednosti ločimo s presledki.

read -n 3 randomname: Dodeljujemo vrednost spremenlivke randomname. Imela bo največ 3 znake. Sistem se po vpisu treh znakov preneha čakati in se vrne v normalno stanje.

set | grep x: izpiše vrednost spremenljivke x

unset x: odstranimo spremenljivko x

source "imeskripte": executa skripto tudi če ni dovoljenja (preveri še). zadeva zelo uporabna, ker na ta način lahko rezultat executane skripte uporabimo v drugi skripti. Na nek način include iz programiranja.

export "ime\_obstoječe\_spremenljivke": exporta spremenljivko tako, da se pojavi v vsakem novem odprtem bash okolju

### SHELL EXPANSIONS - Brace expansion

Primer:

touch file{1,2,3}: ustvari 3 fajle z imeni file1, file2, file3. Brace expansion deluje tudi na mapah, rm komandi...

touch file{1,2,3}test: ustvari 3 fajle z imeni file1test, file2test, file3test

mkdir mapa{1,2,3}{1,2}: ustvari mape z imeni mapa11, mapa12, mapa21, mapa22, mapa31, mapa32

rm -rf file{1,2,3}: zbriše samo te mape

touch ~/mapa1/file{4,5,6}: ustvari fajle z imeni file1, file2, file3 v /home/mapa1, ne glede na to kje se nahajamo

### ALIASES

alias: izpiše vse trenutne aliase

alias ls='ls -l': nadomesti komando ls z ls -l. Pozor tukaj se uporablja enojni narekovaj in ne altgr + 7 sranje.

unalias ls: odstrani ls alias

### EXIT STATUS

Exit status je uspešnost izvedene komande, ki se shrani v spremenljivko $?. Vrednosti so od 0-255. Če je 0 je komanda uspešna in če je katerakoli druga številka, je komanda neuspešna. V skripti je to uporabno npr. da določena mapa mora obstajati in če ne, se skripta ne bo izvršila. Tako v skripti najprej izvedemo cd do mape, nato preverimo $? in gremo dalje, ali če je ni, izpišemo (echo) težavo. V momentu, ko se echo uspešno izvede pa je potrebno vedeti, da se zaradi uspešne izvedbe exit status spremeni v 0. Če je komanda neuspešna ima $? vrednost, ki je odvisna od error levela. Npr. komanda ki ne obstaja vrne 127. Če je komanda pravilna in le argument napačen, vrne 1. 126 je v primeru, da se komanda ni izvedla zaradi problemov s pravicami. Exit statusu pa lahko dodelimo tudi poljubno vrednost. Recimo v skripti, v kakšnem if stavku, če je tak pogoj izpolnjen vrne poljuben exit status in ga zapiše v log fajl.

### COMMAND CHAINING AND COMMAND LISTS

pwd; echo hello world: dve komandi ločeni z delimiterjem semicolon. vsaka izvedena v svoji vrstici. Komand je lahko neomejeno. Komande se zaganjajo ena za drugo.

***Primeri:***

ls -l && pwd: AND. pwd se izvede samo če bo ls -l vrnil exit status 0, torej se izvedel uspešno

ls -l || echo you are now in $PWD: OR. Če ls -l vrne exit status 0, kar ga, se echo ne izvede. Če ls -l ne bi vrnil exit statusa 0, bi se echo izvršil

grep: parser, ki išče določen string in tam kjer ga najde, izpiše celo vrstico

### I\O REDIRECTION

***Primeri:***

cat < helloworld.txt: preusmeri vsebino fajla helloworld.txt na standart output, torej monitor. Znak < tu ni potreben a je dobro, da se ga zapomne zaradi redirectov.

grep Hello < helloworld.txt: Izpiše vrstico iz fajla helloworld.txt, kjer se nahaja string Hello

cat helloworld.txt > helloworld2.txt: Vsebino fajla helloworld.txt preusmeri v fajl helloworld2.txt. Če fajl helloworld2.txt ne obstaja, ga ustvari. Če obstaja, vsebino overwrita.

cat helloworld.txt >> helloworld2.txt: vsebino fajla helloworld.txt prenese v fajl helloworld2.txt. Če fajl helloworld2.txt ne obstaja, ga ustvari. Če obstaja, vsebino helloworld.txt samo doda pod obstoječo vsebino helloworld2.txt (append).

grep Here < helloworld.txt >> helloworld4.txt: iz fajla helloworld.txt sparsamo string Here. Celotno vrstico, kjer se string Here nahaja, appendamo v fajl helloworld4.txt.

./skripta.sh 2> skripta.errors: Standard error output. Vse errorje, ki se bodo pojavili v precesiranju kode skripte outputaj v fajl skripta.errors.

### PIPES

***Primeri:***

ls -lAh | wc -l: zlistamo vsebino trenutnega folderja. Rezultat tega pošljemo word counterju, naj prešteje število vrstic (da je število vrstic pravilno, smo dodali argument -A v ls).

ls -l | sort -r: vsebino ls -l sortiramo reverse

cat data | cut -f 2 ' ': Rezultat komande cat obdelamo tako, da iz fajla imenovanega data vzamemo ven samo drugi stolpec/field (-f). Fieldi so ločeni s presledkom, zato mu moramo povedati še, kakšen delimiter stolpci uporabljajo. Dodamo argument (-d) in v enojne narekovaje vpišemo delimiter, v našem primeru' ', kar je presledek.

### COMMAND SUBSTITUTIONS

***Primeri:***

ls -l `cat lspath.txt`: izpiše vsebino mape ki je navedena v fajlu lspath.txt. Pot do mapeje edina stvar, ki je napisana v fajlu lspath.txt. Uporabljajo se obratni narekovaji altg + 7.

ls -l `cat lspath.txt` > etclisting.txt: isto kot zgornji primer, le da rezultat namesto da ga prikaže na zaslonu, ga da v fajl etchlisting.txt

imespremenljivke=`ls -lAh /etc`: ls mape /etc se bo shranil v spremenljivko "imespremenljivke". Spremenljivko lahko sedaj izpišemo z echo $imespremenljivke. Izpiše se nam vsebina ampak v zelo neurejenem formatu. Da zadevo uredimo damo narekovaje echo "$imesmpremenljivke"

imespremenljivke=$(ls -lAh /etc): enaka komanda kot zgornja, napisana na drug način

imespremenljivke=`netstat -ant | grep 22 | wc -l`: Pove pač recimo za ssh če teče na sistemu. Če je spremenljivka večja kot nič, potem teče. In zadevo nato uporabimo v funkciji v skripti.

imespremenljivke=`< list.txt`: vsebino list.txt spravi v spremenljivko

### QUOTING NUANCES

\: Backslash. Znak, ki sledi \, bo izpisan dobesedno.

'': strong quotes (enojni narekovaji).

"" (dvojni narekovaji): not so strong quotes. Vse kar je znotraj doublequotih se ohrani razen recimo znaka $, še posebej če mu za tem sledi še neka vrednost.

***Primeri:***

echo ti si $lolermao: izpiše samo "ti si", zato ker bash išče spremenljivko $lolermao, zato spodnji primer

echo ti si \$lolermao: izpiše se dobesedno, torej "ti si $lolermao"

mkdir velka muda: ustvari 2 mapi, eno "velka" in drugo "muda"

mkdir velka\ muda: ustvari eno mapo z imenom "velka muda"

echo "this is a test $i": izpiše se text "this is a test" in zatem še vrednost spremenljivke $5. Če te spremenljivke ni, se nič razen "this is a test2 ne izpiše.

echo 'this is a test $i \': vse se izpiše dobesedno, torej "this is a test $i \"

echo "this is a test \\": izpiše se "this is a test \", zato ker smo escapali backslash z backslashom

### HELLO WORLD

Skripto lahko spremenimo v executable na dva načina. Zaženemo jo bodisi s "source", ali pa ga flagamo kot executable

Primer:

source helloworld.sh: pognana skripta s source

chmod +x "imeskripte": spremeni skripto v executable. Zaženejo jo lahko vsi

chmod u+x "imeskripte: samo use in root lahl zaženeta skripto

./imeskripte | wc -l: rezultat skripte "imeskripte" pošlji word counterju naj prešteje vrstice

head -n 1 helloworld.sh: izpiše orvi vrstico skripte helloworld.sh

#!/bin/bash

clear #počisti vse z zaslona

echo What is your name? #izpiše tekst

read name #spremenljivki $name dodeli vrednost, ki bo vtipkana

echo Hello $name #izpiše Hello in vrednost, ki je bila prej vtipkana

#in je postala spremenljivka $name

#includanje. V originalno skripto smo includali/sourcali helloworld.sh.

#helloworld.sh ne rabi #!/bin/bash

. helloworld.sh #executa skripto helloworld.sh

source helloworld.sh #drugi način executanja helloworld.sh

./helloworld.sh | mail -s "Testing" root: Pošlje mail uporabniku root s subjectom "Testing"

mutt: preverimo maile

### FUNCTIONS:

#!/bin/bash

# Date: 13.8.2013

# Author: Dragan

# Purpose: Function Definition

# Created: 13.8.2013

# Modified:

# Definiramo funkcijo z imenom "showdate". "function" ni potreben, dokler

# na koncu imena "showdate" damo "()". Vedno funkcije najprej definiramo,

# nato poklicemo.

function showdate() {

date +%F

}

# Definiramo drugo funkcijo

function showtime() {

date +%r

}

# Definiramo tretjo funkcijo "mailadmin", ki poslje "success" text,

# uporabniku root s subjectom "Succesfull Execution of script".

function mailadmin() {

echo success | mail -s "Succesfull Execution of script" root

}

# Dodamo spremenljivke v funkcijo imenovano "getuserinfo".

# Spremenljivke "firsname" in "lastname" soznotraj funkcije torej so

# lokalne spremenljivke in se uporabljajo samo v tej funkciji.

# Spremenljivke zunaj funkcije, se lahko uporabljajo po celi skripti.

function getuserinfo() {

clear

echo Please enter your firstname and lastname

read firstname lastname

echo Hello $firstname $lastname

}

# Funkcije "showdate", "showtime", "success", "getuserinfo" poklicemo.

# Vrstni red klicanja funkcij ni treba, da je enak, kot vrstni red definiranja

showtime

showdate

getuserinfo

mailadmin

#END

### FOOR LOOPS:

Default delovanje loopa je tako, da gre loop skozi vsak element spiska in ta element dodeli trenutnemu argumentu. Elementi so lahko stringi ali numerični znaki. Argument je spremenljivka. Ta argument ima vsak loop drugo vrednost (vrednost prve vrstice, druge vrstice...).

for arg in [list]; do action item done: klasična for loop sintaksa

let "COUNT +=1" # Vsak cikel povecaj spremenljivko COUNT za 1.

***Primer:***

#!/bin/bash

# Author: Dragan

# Date: 13.8.2013

#Vsaka drzava se steje kot locen element, locen database entry, druga vrstica.

for countries in USA Australia France

do

echo $countries

done

#END

#!/bin/bash

# Author: Dragan

# Date: 13.8.2013

DIR="/etc" # Spremenljivka DIR ima vrednost "/etc".

for file in `ls -A $DIR`

do

echo $file

done

#END

#!/bin/bash

# Author: Dragan

# Date: 13.8.2013

for num in `seq 1 100` # seq 1 100 naredi output list stevilk 1-100.

do

echo The current sequence number is $num

done

#END

#!/bin/bash

# Author: Dragan

# Date: 13.8.2013

$PASSFILE="/etc/passwd" # Vedno je dobro v skripti pot dati v spremenljivko.

# Iz spiska izrezi prvi field z delimiterjem : (v enojnih narekovajih).

for user in `cat $PASSFILE | cut -f 1 -d ':'`

do

echo $user

done

#END

#!/bin/bash

# Author: Dragan

# Date: 13.8.2013

PASSFILE="/etc/passwd" # Vedno je dobro v skripti pot dati v spremenljivko.

COUNT=0 # Spremenljivka COUNT je 0. To bo nas stevec, ki bo vsak loop dodan +1

# Iz spiska izrezi prvi field z delimiterjem : (v enojnih narekovajih).

for user in `cat $PASSFILE | cut -f 1 -d ':'`

do

echo $user

let "COUNT +=1" # Vsak cikel povecaj COUNT za 1.

done

# Izpisemo koncno vrednost spremenljivke COUNT, zato izven for loopa.

# Vrednost spremenljivke COUNT pove stevilo loopov, torej vrstic/elementov.

echo There are $COUNT users registered on the system.

#END

### WHILE LOOPS

While loop testira zgornji del loopa (deskriptor loopa, torej vsebina v oglatih oklepajih) za exit statusi 0. Če je exit status 0, potem se izvede koda v bodyju while loopa. Ko exit status ni več 0 se zanka zaključi/exita. Če exit status nikoli ni 0, se loop nikoli izvede. To pa ne pomeni, da se skripta ni izvedla pravilno. Skripta se seveda nadaljuje. Long story short, dokler je zgornji del TRUE, izvajaj. Ko je NOT TRUE, ustavi.

***Primeri:***

#!/bin/bash

# Author: Dragan

# Date: 18.8.2004

# Dolocimo spremenljivki NUM in MAX. To bo nas min in max.

NUM=0

MAX=20

while [ "$NUM" -lt "$MAX" ] # Dokler je $NUM manjsi od $MAX...

do

echo $NUM # Izpisi $NUM

let "NUM += 1" # Povecaj $NUM za 1, Let inkrementira/dekrementira vrednosti v skripti.

done

#END

### UNTILL LOOP

Ravno obratno od while loop. Dokler je deskripcija loopa, torej stanje v oglatih oklepajih NOT TRUE, izvajaj. V momentu, ko je TRUE, ustavi.

***Primeri:***

!/bin/bash

# Author: Dragan

# Date: 18.8.2013

NUM=100

MIN=19

until [ "$NUM" -eq "$MIN" ]

do

echo $NUM

let "NUM -= 1"

done

#END

. untilloop1.sh > untilloop1.txt: Output skripte untilloop1.sh se prenese v fajl untilloop.txt. Za zagon skripte smo uporabili ".", kar je identično kot "source".

. untilloop1.sh > /dev/null: Karkoli želi skripta prikazati na zaslonu, ne prikaži. /dev/null je nekakšna črna luknja.

#!/bin/bash

# Author: Dragan

# Date: 18.8.2013

# Neskončna untill zanka. Spremenljivka $STATUS ni enaka 0, zato ker sploh ni dodeljena. To pomeni, da pogoj v oglatih oklepajih nikoli ni izpolnjen. Untill loop pa se izvaja takrat, ko pogoj ni izpolnjen, torej v tem primeru vedno.

until [ "$STATUS" -eq "0" ]

do

ping -c 1 192.168.1.35 # -c 1 pomeni, da na ta naslov pošljemo 1 paket.

echo The host is down

done

#END

#!/bin/bash

# Author: Dragan

# Date: 18.8.2013

# Neskončna untill zanka. Spremenljivka $STATUS ni enaka 0, zato ker sploh ni dodeljena. To pomeni, da pogoj v oglatih oklepajih nikoli ni izpolnjen. Untill loop pa se izvaja takrat, ko pogoj ni izpolnjen, torej v tem primeru vedno.

until [ "$STATUS" -eq "0" ]

do

ping -c 1 192.168.1.35 # -c 1 pomeni, da na ta naslov pošljemo 1 paket.

echo The host is down

done

#END

### CONTROL STRUCTURES

V skripti, po tem ko vpišeš "if", se dejansko izvede test zadeve v oglatih oklepajih (test kot built in komanda, opisana v tem notepadu). Testiramo lahko stringe ali numerične znake. Za različne vrednosti (stringi, numeric...) moramo uporabljati tudi različne operatorje (-eq, =). Za numerične znake je idealno da se uporabljajo -eq... Za besede je primerno, da se uporablja dejansko operator torej = ...

Primeri:

test 2 -eq 2: Test numeričnih znakov. Pri tem bo exit status (echo $?) seveda 0, ker 2 je enako kot 2.

test linux = linux\_cbt: V tem primeru testiramo besede, zato uporabljen operator =. Exit status (echo $?) seveda 1.

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

# Funkcija izpiše "Both values are equal, ker je 1 enako 1, torej exit status $? zadeve v oglatih oklepajih 0.

if [ 1 -eq 1 ]

then

echo Both values are equal

else

echo Both values are unequal

fi # Tako vedno koncamo if funkcijo

#END

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

if [ "linuxcbt\_scripting" = "linuxcbt" ] # Primer dveh stringov kjer je operand =.

then

echo Both values are equal

else

echo Both values are unequal

fi # Tako vedno koncamo if funkcijo

#END

#!/bin/bash

# If statement nestan v for loop. Ta loop smo uporabili že prej in je v tem notepadu. Kaj to naredi je, da ko ima $countries vrednost USA, izpiše welcome to the USA. Ko ima $countries vrednost Jamaica, izpiše One love. Ko nima nobene izmed teh dveh vrednosti, preprosto izpiše državo v novi vrstici.

for countries in USA Australia France Latvia Argentina Jamaica

do

if [ "$countries" = "USA" ]

then

echo Welcome to the $countries

elif [ "$countries" = "Jamaica" ]

then

echo One Love

else

echo $countries

fi

done

#END

### CONTROL STRUCTURES 2:

Argumenti: -e (preveri če file/direktorij obstaja), -d (preveri, če direktorij obstaja), -nt (newer than)

***Primeri:***

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

if [ -e forloop1.sh ] # -e argument pomeni "exists", torej če file forloop1.sh obstaja.

then

echo The File Exists

else

echo The file does NOT exist

fi

#END

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

# Enako kot zgornji primer, le da tu ime fajla postavimo v spremenljivko $FILE

FILE="forloop1.sh"

if [ -e $FILE ]

then

echo The File Exists

else

echo The file does NOT exist

fi

#END

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

# Testiramo kateri file, dan v spremenljivko, je novejši.

FILE1="test1"

FILE2="test2"

if [ $FILE1 -nt $FILE2 ]

then

echo $FILE1 is newer

else

echo $FILE2 is newer

fi

#END

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

netstat -ant | grep :80 # Preverimo ce sistem poslusa port 80.

APACHESTATUS="$?" # Spremenljivka APACHESTATUS naj bo enaka kot exit status.

if [ "$APACHESTATUS" -eq 0 ] # Test, ce je vrednos APACHESTATUS 0, potem echo...

then

echo Apache is Up and Running

fi

#END

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

# Preverimo ce sistem poslusa port 80. Output netstata posljemo v /dev/null,

# ki je crna luknja, ker tega nocemo prikazati na zaslonu

netstat -ant | grep :80 > /dev/null

APACHESTATUS="$?" # Spremenljivka APACHESTATUS naj bo enaka kot exit status.

if [ "$APACHESTATUS" -eq 0 ] # Test, ce je vrednos APACHESTATUS 0, potem echo...

then

echo Apache is Up and Running

fi

#END

#!/bin/bash

# Author: Dragan

# Date: 20.8.2013

# Preverimo ce sistem poslusa port 80. Output netstata posljemo v /dev/null,

# ki je crna luknja, ker tega nocemo prikazati na zaslonu

netstat -ant | grep :80 > /dev/null

APACHESTATUS="$?" # Spremenljivka APACHESTATUS naj bo enaka kot exit status.

if [ "$APACHESTATUS" -eq 0 ] # Test, ce je vrednos APACHESTATUS 0, potem echo...

then

echo Apache is Up and Running

# Ce tece apache, preveri delovanje MySQL na portu 3306.

# Tudi ta netstat output posljemo v /dev/null crno luknjo.

netstat -ant | grep 3306 > /dev/null

MYSQLSTATUS="$?" # Spremenljivka MYSQLSTATUS naj dobi vrednost $?.

if [ "$MYSQLSTATUS" != 0 ]

then

echo MySQL is NOT Running

else

echo MySQL is Running

fi

else

echo Apache is NOT Running

fi

#END

### CONTROL STRUCTURES 3

***Primer dolge verzije (prva skripta) in kratke verzija s "case" stavki (druga skripta):***

#!/bin/bash

# Spremenljivka $countries bo vsak loop imela drugo vrednost (državo).

# Za vsako vrednost smo ji dodelili, da se izvrši nek random echo.

for countries in USA Australia France Latvia Argentina Jamaica

do

if [ "$countries" = "USA" ]

then

echo Welcome to the $countries

elif [ "$countries" = "Jamaica" ]

then

echo One Love

elif [ "$countries" = "Australia" ]

then

echo Gooday Mate

elif [ "$countries" = "France" ]

then

echo Merci

elif [ "$countries" = "Latvia" ]

then

echo Welcome to Latvia

elif [ "$countries" = "Argentina" ]

then

echo Buenos Dias

else

echo $countries

fi

done

#END

#!/bin/bash

for countries in USA Australia France Latvia Argentina Jamaica Cuba

do

case $countries in USA ) # Ja, to je zaklepaj in mora biti tako!

echo "Welcome to North America" ;; # Dva podpicja za zakljucek casa.

Australia )

echo "Good Day Mate" ;;

France )

echo "Merci" ;;

Latvia )

echo "Welcome to the USSR" ;;

Argentina )

echo "Buenos Dias" ;;

Jamaica )

echo "One Love" ;;

\* ) # Zvezdica za primer nenavedenih vrednosti spremenljivke $countries (cuba)

echo "Welcome to another land" ;;

esac

done

#END

### POSITIONAL PARAMETERS

Positional parameter je parameter, ki ga napišemo skupaj s komando za exekucijo skripte. Globalna spremenljivka $# nam pove število parametrov, ki smo jih dodali.

V primeru, da sourcamo skripto (. skripta.sh), ki vsebuje v kodi "exit", se nam bo shell okno zaprlo. Da do tega ne pride, je treba skripto označiti za executable in jo executati (./skripta.sh). Zakaj do tega pride? Zato ker, ko skripto executamo z ./, se executa v sub shellu, torej drugem shellu, kot pa tisti v katerega pišemo komande. To mi sicer ne vidimo.

***Primer:***

. skripta1.sh 2 51: Tu smo sourcali skripto in ji dodali 2 parametra, 2 in 51, 2 je prvi parameter in katerega vrednost nosi sedaj spremenljivka $1. Spremenljivka $2 nosi vrednost 51.

#!/bin/bash

# Author: Dragan

# Date: 23.8.2013

# Deklariramo spremenljivko $BADPARAM, za primer, da ni dveh parametrov,

# pri eksekuciji skripte. V tem primeru bomo skripto exitali v if stavku spodaj.

# Exitali jo bomo z error levelom 165. To je tipicna cifra za errrorje te vrste.

# Error level ali exit status. To je isti kurac.

# Spremenljivko uporabljamo zato, da preprosto spremenimo error level, ne da bi spreminjali kodo.

BADPARAM=165

# Ce stevilo parametrov ni enako 2, echo text in exit skripte z error levelom vrednosti # spremenljivke $BADPARAM.

if [ $# != 2 ]

then

echo This script requires 2 arguments

# Sistemska spremenljivka $# pove, koliko parametrov smo uporabili.

echo "You've entered $# parameters"

exit $BADPARAM

fi

# Begin Sequence Command. $1 ima vrednost prvega parametera, in $2 drugega.

seq $1 $2

#END

### SELECT MENUS

Select menu v naših primerih je v bistvu vprašanje, in opcije odgovora, ki so možne za izbiro. Za vsako opcijo, ki jo izberemo, se izvede neka if funkcija. Po defaultu uporablja select ps3 prompt. Default prompt, ki ga mi uporabljamo je ps1.

#!/bin/bash

# Author: Dragan

# Date 23.8.2013

# Purpose: Illustrate basic usage of select

# Osnovni izgled select menuja. Var je lahko nadomeščen z random imenom, ni treba da je var.

select var in "Choice1 Choice2"

do

command

break

done

#END

#!/bin/bash

# Author: Dragan

# Date 23.8.2013

# Purpose: Illustrate basic usage of select

# Text, ki nas poziva k izbiri opcije.

# Brez tega, bi zaradi delovanja ps3 prompta, namesto texta bilo samo: #?

PS3='Select your choice: '

# Imena opcij shranimo v spremenljivko LIST

LIST="Choice1 Choice2"

# Spremenljivka "var" bo dobila vrednost izbrane opcije.

# za različne vrednosti se izvedejo različne operacije.

select var in $LIST

do

if [ $var = "Choice1" ]

then

echo Hello World

elif [ $var = "Choice2" ]

then

echo Goodbye World!

elif [ $var = "Quit" ]

then

# Zapri shell. Če je skripta sourcana, se zapre naš delovni shell.

# Če je skripta executana, se zapre subshell v katerem se je skripta executala.

exit

fi

# Skripta naj se po izbiri opcije prekine (break)

break

done

#END

#!/bin/bash

# Author: Dragan

# Date 23.8.2013

# Purpose: Illustrate basic usage of select

PS3='Select your choice: '

LIST="System Quit"

select var in $LIST

do

if [ $var = "System" ]

then

# Skripta tu namesto prej echo, izvede pregled sistemskega loga.

# Log refresha na 2 sekundi

watch tail /var/log/messages

elif [ $var = "Quit" ]

then

exit

fi

break

done

#END

### MOVE MANY FILES

Bash po defaultu nima funkcije preimenovanja, kopiranja večih fajlov na enkrat. Tu je uporabna skripta za izvajanje teh operacij s pomočjo for loopa. Za boljše razumevanje, še enkrat poglej kako deluje for loop. Drugi primer skripte je fleksibilna skripta in uporablja positional parametre. Za razumevanje poglej, kako delujejo positional parametri.

#!/bin/bash

# Author: Dragan

# Date: 3.10.2013

# Purpose: Used to rename many files

# Spisek for zanke naj bodo vsi fajli ki se začnejo na test\*

for file in `ls -A test\*`

do

# Vsak fajl iz spiska preimenuj v staroimefajla.old

mv $file $file.old

done

#END

#!/bin/bash

# Author: Dragan

# Date: 3.10.2013

# Purpose: Used to rename many files (flexible, with positional parameters)

# V tej skripti bomo vsem fajlom, ki se zacnejo na $1, dodali suffix $2.

# 165 bo nas exit status v primeru neupostevanja stevila positional param.

BADARG=165

# Stevilo potrebnih positional parametrov ob zagonu skripte.

REQPARAM=2

# Preverimo, ce je stevilo positional parametrov, ob zagonu skripte pravilno.

if [ $# != $REQPARAM ]

then

echo You need $REQPARAM positional parameters!

exit $BADARG # Skripto exitamo z exit statusom 165 ($BADARG)

fi

# Spisek for zanke naj bodo vsi fajli, ki se zacnejo na $1 positional param.

# \* je dodana v zanki zato, ker je kot positional parameter pri zagonu skripte

# ne moremo uporabiti. Bash jo namrec uposteva kot locen positional parameter!

for file in `ls -A $1\*`

do

# Vsakemu fajlu iz spiska for loopa, dodaj suffix $2 positional param.

mv $file $file.$2

done

#END

### NETWORK CHECK

Skripta za preverjanje network accessibility. Druga skripta je bolj fleksibilna različica prve. Outpute skript lahko redirectamo v kakšen logfile. Če želimo skripto poganjati samodejno, jo moramo dati v eno izmed cron map (npr. /etc/cron.daily).

***Primeri:***

./networkcheck.sh > pinglog.`date +%F`: Outputamo v log file z imenom pinglog."današnji\_datum"

#!/bin/bash

# Author: Dragan

# Date: 6.10.2013

# Purpose: This script monitors Internet Connectivity

SITE="www.google.com"

# Posljemo 2 paketa na google.com. -c pomeni count.

# Ker rezultata pinga ne rabimo videti, ampak rabimo vedeti, le ce je uspesen,

# ga posljemo v /dev/null.

ping -c 2 $SITE > /dev/null

# Preverimo, ce je exit status 0. Ce ni, potem izpisemo datum in text.

if [ $? != 0 ]

then

echo `date +%F`

echo There seems to be Internet Connectivity issues!

Fi

#END

#!/bin/bash

# Author: Dragan

# Date: 6.10.2013

# Purpose: This script monitors Internet Connectivity

# Preveri, ce je bil positional parameter vnesen. Ce je bil vnesen, $SITE dobi

# vrednost positional parametra. Ce ne, $SITE ostane www.google.com.

if [ "$#" -eq "0" ]

then

SITE="www.google.com"

else

SITE="$1"

fi

# Posljemo 2 paketa na $SITE. -c pomeni count.

# Ker rezultata pinga ne rabimo videti, ampak rabimo vedeti, le ce je uspesen,

# ga posljemo v /dev/null.

ping -c 2 $SITE > /dev/null

# Preverimo, ce je exit status 0. Ce ni, potem izpisemo datum in text.

if [ $? != 0 ]

then

echo `date +%F\ %r`

echo There seems to be Internet Connectivity issues!

cat log.`date +%F` | mail -s "There is an issue with connectivity!" root

fi

#END

### FILE DIFFERENCES

Skripta, ki opazuje spremembe fajlov v folderjih. Najprej administrator naredi snapshot folderja, ki ga želo opazovati (ls -A > ~/snapshots/filelist1). Ta originalni snapshot bo referenca. Snapshot file in skripto shranimo v ločeno mapo od tiste, katere snapshot smo naredili (opazovana mapa). Da bi gledali spremembe fajlov v folderju, primerjamo originalni snapshot z novim snapshotom, s komando "diff".

***Primeri:***

diff filelist1 filelist2 | cut -f 2 -d ' ': Komanda diff za opazovanje razlik v fajlih. V našem primeru primerjamo dva snapshota. Output izrežemo tako, da vzamemo samo field 2 z delimiterjem ' ', saj je prvi field puščica, ki nam nič ne pomaga (<).

Rezultat tega bi izgledal tako: 3d2 file1

Pri tem je "file1" tisti fajl, ki ga ni v originalnem snapshotu. Tisti string spredaj (3d2) niti ne vem kaj je. Pojavlja se tudi v outputu prve skripte spodaj, a ga bomo v drugi skripti odstranili.

#!/bin/bash

# Author: Dragan

# Date: 7.10.2013

# Purpose: Monitor directory for file changes

# Opazovan folder smo dali v spremenljivko.

# Ko sem hotel uporabiti namesto celotne poti, tilde expansion ~/bash\_l2p,

# je javljalo, da mapa ne obstaja. Ko sem odstranil narekovaje, je delalo.

# Narekovaje sem zdaj pustil, ampak preveri zakaj je jebalo.

MONITORDIR="/home/pi/bash\_l2p"

# Spisek fajlov v opazovanem folderju outputamo v fajl filelist2, ki se bo

# samodejno ustvaril v mapi, kjer se nahaja skripta.

ls -A $MONITORDIR > ~/l2p/filelist2

# Primerjamo originalni snapshot (filelist1), s snapshotom, ki ga je ustvarila

# skripta (filelist2). Skripta in snapshoti naj bodo v locenem folderju!

# $FILEDIFF dobi vrednost te primerjave.

FILEDIFF=`diff filelist1 filelist2 | cut -f 2 -d ' '`

# Izpišemo razlike med snapshotoma.

echo $FILEDIFF

#END

#!/bin/bash

# Author: Dragan

# Date: 7.10.2013

# Purpose: Monitor directory for file changes

# Opazovan folder smo dali v spremenljivko.

# Ko sem hotel uporabiti namesto celotne poti, tilde expansion ~/bash\_l2p,

# je javljalo, da mapa ne obstaja. Ko sem odstranil narekovaje, je delalo.

# Narekovaje sem zdaj pustil, ampak preveri zakaj je jebalo.

MONITORDIR="/home/pi/bash\_l2p"

# Spisek fajlov v opazovanem folderju outputamo v fajl filelist2, ki se bo

# samodejno ustvaril v mapi, kjer se nahaja skripta, ce mu ne navedemo poti.

ls -A $MONITORDIR > ~/l2p/filelist2

# Primerjamo originalni snapshot (filelist1), s snapshotom, ki ga je ustvarila

# skripta (filelist2). Skripta in snapshoti naj bodo v locenem folderju!

FILEDIFF=`diff filelist1 filelist2 | cut -f 2 -d ' '`

### MONITOR SERVICE

Skripta za opazovanje določenega servica na sistemu. Rabimo v primeru, da imamo kakšen kritičen service, ki ga bo skripta preverjala če teče. Če ne, ga bo poskušala nekajkrat restartati. V našem primeru bomo opazovali MySQL service, ki se veže na port 3306. Torej, če želimo opazovati, da ima MySQL network connectivity, gledamo ta port. Če bi želeli preveriti, ali service samo teče, ga lahko preverimo z built-in komando "service", ali pa pogledamo v mapo /var/run, kjer večina daemonov kreira svoje log fajle. Nujno poglej, kako deluje while loop, preden to skripto bereš.

#!/bin/bash

# Author: Dragan

# Date: 10.10.2013

# Preverjanje, ce je TCP port 3306 v uporabi s strani MySQL serverja

netstat -ant | grep 3306 > /dev/null # netstat s switchem -ant pokaze SAMO TCP

# porte 3306 v uporabi.

# Output posljemo v null praznino.

MYSQLSTATUS=`echo $?` # $MYSQLSTATUS dobi vrednost exit statusa.

SERVICENAME="mysql" # Service, ki ga bomo preverjali.

# $COUNT in $THRESHOLD sta spremenljivki za while loop. Zelimo, da se loop

# izvede 3x (od 0 do 2)

COUNT=0

THRESHOLD=2

if [ $MYSQLSTATUS != 0 ] # Ce $MYSQLSTATUS ni 0, torej ce MYSQL ni povezljiv..

then # Ce ni povezljiv, izvedi ta while loop. Izvede se kot receno 3x.

while [ $COUNT -le $THRESHOLD ] # Dokler je $COUNT <= $THRESHOLD...

do

service $SERVICENAME start # Zazeni service (mysqld).

if [ $? != 0 ] # Ce novi exit status takoj po startu mysqld

# ni 0 torej se mysqld ni mogel startati...

then

let "COUNT +=1" # Povecamo $COUNT za 1.

# Drugace neskonci loop.

else

# Ce pa je bil novi exit status 0, izpisi success,

# in zapusti skripto z exit statusom 0.

echo $SERVICENAME has been started successfully

exit 0

fi

done

# Ce se je while loop izvedla dokonca, in ni uspela zagnati mysqld

# v treh poskusih, poslji mail administratorju.

echo Problems starting $SERVICENAME | mail -s "$SERVICENAME Failure"

else

# Ce je bil ze ob zagonu skripte $MYSQLSTATUS 0, potem izpisi....

echo $SERVICENAME is already running

fi

#END

### FTP SYNCHRONISATION

Skripte za sinhronizacijo vsebine med dvema hostoma preko LFTP mirror funkcionalnosti. LFTP podpira veliko protokolov prenosa, vključno s FTP, SFTP, HTTP/HTTPS. LFTP ima funkcijo imenovano "mirroring", ki lahko mirrora in reverse mirrora fajle iz lokalnega sistema na oddaljenega in obratno. Naša skripta bo sestavljena iz glavne skripte in subfajla (includana skripta). Glavna skripta bo v določenem trenutku klicala (includala) subfile, iz katerega bo prebrala komande za upravljanje z LFTP. Pri subfajlu ni potrebno na začetku napisati #!/bin/bash.

Ponavadi se take skripte zaganjajo v časovnih intervalih. Tako recimo lahko glavno skripto postavimo v /etc/cron.hourly/ in skripta se bo poganjala vsako uro.

***Primeri:***

lftp pi@192.168.1.9: Povezava lftp na 192.168.1.9, kot user pi, preko FTP protokola (FTP je default)

lftp sftp://pi@192.168.1.9: Povezava lftp na 192.168.1.9, kot user pi, preko SFTP protokola.

open sftp://pi@192.168.1.9: Drugi način povezave. Pred tem smo v shell vpisali lftp, da se nam je odprla lftp konzola.

open -upi,mojegeslo sftp://192.168.1.9: Tretji način povezave. V lftp konzolo smo za "open" dali switch -u, ki omogoča direkten vnos usernama in passworda v komando.

lftp -f imefajla: To je komanda, ki odpre konzolo lftp. Switch -f pomeni, da naj komande v konzolo vnaša iz specifičnega fajla, v našem imenu "imefajla".

Primeri za lftp konzolo:

!ls: Ko smo že vzpostavili povezavo lftp, nam predznak "!" pomeni, da komando "ls" izvajamo na lokalnem sistemu. Brez predznaka "!", bi dobili spisek fajlov na oddaljenem sistemu, na katerega smo z lftp povezani.

lcd: spremeni direktorij na lokalnem sistemu

mirror: Vsi fajli v trenutnem remote folderju bodo poslani na lokalni sistem (-R reverse mirror, -n samo novejši fajli bodo poslani...)

#!/bin/bash

# Author: Dragan

# Date: 11.10.2013

# Purpose: To synchronize folders between disparate hosts (MAIN SCRIPT)

# $SCRIPTHOME vsebuje pot do home folderja,

# kjer se nahaja nasa skripta za includanje.

SCRIPTHOME="/home/pi/bash\_l2p"

# Nasa skripta za includanje.

LFTPSCRIPT=$SCRIPTHOME/ftpsynchronize.lftp

# Zagon lftp, s switchem -f, ki pomeni, da bodo komande v lftp vnesene preko

# locene (includane) skripte ftpsynchronize.lftp.

lftp -f $LFTPSCRIPT

# Purpose: Subfile, ki ga bo glavna skripta klicala.

# Purpose: Vsebuje nekaj vrstic komand, ki bodo upravljale lftp.

# -u switch omogoca direkten vnos passworda v komando za ustvarjanje povezave.

open -udragan,bionicm4n sftp://192.168.1.102

# Pojdi v bash\_l2p folder na REMOTE hostu.

cd /home/dragan/bash\_l2p

# Pojdi v bash\_l2p folder na LOCAL hostu.

lcd /home/pi/bash\_l2p

# NOVE fajle, ki se pojavijo na local hostu, uploadaj na remote host (-R. -n).

mirror -Rn

### PARSE LOGS

Skripta za parsanje logov. Probal bom sparsati /var/log/user.log, kjer inadyn logira svoj activity. Namen bo, da bomo imeli omejitev števila errorjev v logu. Ko je ta omejitev presežena, bo skripta izvedla, kar bo pač izvedla. Ključna beseda, ki jo bom iskal v logu bo pač "Error", ki se pojavlja, kadar inadyn naleti na napako.

***Primeri:***

awk '/Error/ { print $1, $2, $3, $9 }' /var/log/user.log.1: Podobno kot kombinacija grep in cut. awk v fajlu /var/log/user.log.1 poišče vrstico z besedo "Error" ter iz te vrstice izpiše stolpce 1, 2, 3, 9. Znak / v besedi Error je tam kot escape character.

#!/bin/bash

# Author: Dragan

# Date: 15.10.2013

# Purpose: Parse Service Logs (user.log poskus)

# Fajl iz katerega beremo loge, dan v spremenljivko.

LOGFILE="/var/log/user.log.1"

# Kljucni string v logu, ki ka iscemo.

BADSTRING="Error"

# Iz fajla user.log.1, v vrsticah kjer se pojavlja string "Error", izpisi

# samo 1., 2., 3. in 9. stolpec. Tokrat smo grep output poslali v awk namesto

# cut, ki smo ga uporabljali do zdaj. awk je napreden parser in dejansko ne

# potrebuje grep, a je tu vseeno notri.

grep $BADSTRING $LOGFILE | awk '{ print $1,$2,$3,$9 }'

#END

#!/bin/bash

# Author: Dragan

# Date: 15.10.2013

# Purpose: Parse Service Logs (user.log poskus). Ce se error pojavi veckrat,

# kot je vrednost positional parametra, poslji mail uporabniku Pi.

# Fajl iz katerega parsamo kljucno besedo.

LOGFILE="/var/log/user.log.1"

# Kljucna beseda.

BADSTRING="Error "

# Preveri, ce smo vnesli 1 positional parameter.

if [ $# != 1 ]

then

echo At least 1 parameter is required for the Treshold Value

fi

# Limit, kolikokrat se lahko ponovi kljucna beseda v fajlu. Limit je

# positional parameter $1.

THRESHOLD=$1

# Sparsaj besedo $BADSTRING z awk parserjem iz fajla $LOGFILE.

# Tu verjetno tisti print in stolpci niso potrebni, saj ne potrebujemo izpisa

# parsa, ampak le stevilo vrsticv fajlu, v katerih se $BADSTRING pojavlja.

OFFENSES=`awk '$BADSTRING { print $1,$2,$3,$9 }' $LOGFILE | wc -l`

# Ce bo stevilo vrstic, v katerih se beseda Error pojavja, presega vrednost

# spremenljivke $THRESHOLD, izpisi text, in ga poslji po mailu pi userju.

# V RPi, sentmail komponente nima, tako da mail ne dela.

if [ $OFFENSES -gt $THRESHOLD ]

then

echo $OFFENSES inadyn errorjev. | mail -s "Inadyn errorji!" pi

fi

#END

### BACKUP FILES 1

Skripta, ki bo backupirala specifičen folder, zipala vsebino (.tar, gzip) in uporabila SCP protokol, da bo fajl premaknila na oddaljen host. Skripta bo pred prenosom backupa na remote host zahtevala password remote hosta.

#!/bin/bash

# Author: Dragan

# Date: 19.10.2013

# Purpose: Used to backup files and/or directories locally and store remotely

# Folder kjer bomo shranjevali backupe.

BACKUPDIR=~/backup

# Folderja, ki ju bomo backupirali. Ker sta dva folderja, smo morali uporabiti

# narekovaje (doublequotes). V njih pa tilde expansion ne deluje.

# Nadomestili smo ga z $HOME.

SCRIPTDIR="$HOME/bash\_l2p $HOME/testfolder"

# Ime backup fajla. To je celoten backup zapakiran v 1 fajl in zazipan.

# Ce hocemo se datum v imenu fajla, moramo uporabiti command substitution ``.

# Dobro ce mu damo koncnico .tgz, zato da uporabnik ve, da je bil fajl .tar-an.

BACKUPFILE=scripts.backup.`date +%F`.bz2

# Host na katerega bomo backup uploadali.

BACKUPHOST=dragan@192.168.1.102:/home/dragan/

# Nas limit, koliko backupov naj se naredi.

THRESHOLD=7

# Definiramo prvo funkcijo v delu skripte, ki preverja obstoj backup folderja.

function checkbackupdir() {

# Skripta preveri, ce mapa z imenom vrednosti $BACKUPDIR obstaja.

if [ ! -e $BACKUPDIR ]

then

# Ce ne obstaja, mapo ustvari in dodeli vrednost $COUNT=0.

# Saj ce ni mape, potem tudi nobenega obstojecega backupa ni.

echo Creating Backup Directory because it doesn\'t exist!

COUNT=0

mkdir ~/backup

else

# Ce obstaja, prestej stevilo obstojecih backupov v mapi #BACKUPDIR.

COUNT=`ls $BACKUPDIR/scripts.\* | wc -l`

fi

}

# Definiramo drugo funkcijo v delu skripte, ki je zadolzen za izvedbo backupa.

function backup() {

# Ce he stevilo trenutnih backupov manjse ali enako 7...

if [ $COUNT -le $THRESHOLD ]

then

# V folderju $BACKUPDIR bomo naredili fajl z imenom $BACKUPFILE.

# Vsebina tega fajla je zapakiran in zazipan folder $SCRIPTDIR.

# Output posljemo v null, da ne gledamo, kaj sistem zipa (.tar-a).

tar -cjvf $BACKUPDIR/$BACKUPFILE $SCRIPTDIR > /dev/null

# Preveri ce se je fajl $BACKUPFILE uspesno kreiral.

# if stavek je tokrat napisan v eni vrstici, tako malo za spremembo.

if [ $? != 0 ]; then echo Problems Creating Backup File; fi

# Skopiraj narejen backup, preko SCP protokola na $BACKUPHOST.

scp $BACKUPDIR/$BACKUPFILE $BACKUPHOST

# Preveri, ce se je $BACKUPFILE uspesno prekopiral na $BACKUPHOST.

if [ $? != 0 ]; then echo Problems Copying Backup File to Backup Host; fi

fi

}

# Obe prej deklariranu funkciji poklicemo.

checkbackupdir

backup

#END

### LOGGING TECHNIQUES

Output skripte bomo zalogirali na način, kot izgleda sys log daemonov (/var/log). Ta emulacija sys logov bo naredila naše loge lažje za branje in razumevanje.

Logi v linuxu se ponavadi rotirajo (log rotation - arhivirajo kot jast razumem). Tisti log, ki nima končnice, je ponavadi naš trenutni aktivni log.

Logi izgledajo ponavadi tako: Ime meseca, dan v mesecu, Timestamp (24h), Hostname, ime daemona (aplikacije), sporočilo, ki se echoja.

V našem primeru bomo ustvarili skripto, ki bo kreirala log fajl, s standardno obliko. Imena log fajlov, ki jih bo skripta ustvarila bodo različna (preveri komentarje skript). To skripto bomo potem includali na konec eno izmed že obstoječih skript. Če se bo obstoječa skripta uspešno izvršila do točke, kjer bomo includali log skripto (preverili bomo $?), potem bomo sourcali log skripto s pozicijskim parametrom 0. To bo pomenilo, da se bo v log fajl zapisal "Success". V nasprotnem primeru bo sporočilo "Failure".

***Primeri vsebine log fajla:***

Oct 20 06:41:18 raspberrypi rsyslogd: [origin software="rsyslogd" swVersion="5.8.11" x-pid="1858" x-info="http://www.rsyslog.com"] rsyslogd was HUPed

#!/bin/bash

# Author: Dragan

# Date: 23.10.2013

# Name: logscript log1.sh

# Purpose: To illustrate Syslog-like logging from our scripts

# Script can be used standalone, but it will be used as included script to

# an already existing script.

# Prikaze datum v formatu, kot ga uporablja syslog.

mydate=`date +%b\ %d\ %T`

# Ime hosta

myhost=`hostname`

# Ime folderja, v katerem se skripta nahaja.

mydirname=`dirname $0`

# Ime skripte. Da to dela, mora biti skripta executable. Preveri zakaj.

myscript=`basename $0`

# To bo nase sporocilo v logu. Vrednost $mymessage bo odvisna od pozicijskega

# parametra ob zagonu skripte. Ce pozicijski parameter 0, potem se bo v log

# zapisalo sporocilo "Success". Ce ne, potem "Failure".

if [ $1 -eq 0 ]

then

mymessage="Success"

else

mymessage="Failure"

fi

# Echo vrednosti zgornjih spremenljivk in output appendaj v fajl z imenom

# trenutnega datuma + imena skripte, ki jo poganjamo + .log.

# Ime skripte zato, da ko to skripto includamo v obstojeco, loge locimo med sabo.

echo $mydate $myhost $mydirname $myscript $mymessage >> `date +%F.$myscript.log`

#END

#!/bin/bash

# Author: Dragan

# Date: 29.10.2013

# Primer includanja skripte za logiranje, v ze obstojeco skripto

# Del obstojece skripte.

if [ "linuxcbt\_scripting" = "linuxcbt" ]

then

echo Both values are equal

else

echo Both values are unequal

fi

# Del, kjer smo includali naso skripto za izdelovanje logov log1.sh.

# Ce je exit status skripte do tukaj 0, torej uspesno izvedena koda, potem

# sourcaj skripto log1.sh s pozicijskim parametrom 0. Ce exit status ni 0

# potem log1.sh sourcaj s pozicijskim parametrom 1, kar bo v log fajl zapisalo

# failure. Glej logscript za izdelovanje log fajlov, kako do tega pride.

if [ $? -eq 0 ]

then

. log1.sh 0

else

. log1.sh 1

fi

#END

### LOG ABSTRACT

Naredili bomo log abstraction skripto, ki bo parsala vse (več) loge v /var/log za določenimi entryji (za določen datum) in jih shranila v ločen fajl, ki bi ga kasneje pregledali.

Skripta bo torej morala omogočati procesiranje večih log fajlov ter pregledati vnose vrstico po vrstico in najti vrstice, ki se nanašajo na trenutni datum.

Kot zanimivost, načeloma tisti daemoni, ki ne ustvarijo log fajla za sebe, shranijo svoje loge v messages.log.

V Linux sistemu, vsakič ko se izvede log rotacija (cron), se najprej ustvari log fajl. Ta je na začetku prazen in ni nujno, da se bo sploh kdaj napolnil. Take fajle moramo poskrbeti, da jih ne procesiramo za brez veze.

#!/bin/bash

# Author: Dragan

# Date: 30.10.2013

# Illustrate how to abstract daily log entries

# Trenutni datum (mesec, dan), v taki obliki, kot se nahaja v log fajlih.

mydate=`date +%b\ %d`

# Ime skripte, ki jo poganjamo.

myscript=`basename $0`

# Log fajl, ki jih procesiramo. Spremenljivke smo poimenovali log1, log2 zato,

# ker bo na tak nacin lazje izvesti for zanko, ki bo sla skozi vse loge.

log1=/var/log/messages

log2=/var/log/mail.log

log3=/var/log/mysql.log

log4=/var/log/secure

log5=/var/log/auth.log

# Argument log v for loopu bo prvi loop dobil vrednost spremenljivke $log1.

# If stavek bo nato preveril, ce je vrednost $log dejansko obstojec fajl in da

# ta fajl ni prazen fajl (ni zerobyte fajl). Ko sta oba pogoja izpolnjena, se

# vrednost $log nato echoja (/var/log/messages)in doda zraven se text BEGIN.

# Iz tega fajla bo nato grep sparsal vse entryje iz dneva zagona skripte "$mydate".

# V primeru, da bi se pri grepu pojavil kaksen error, ga bomo standard error

# outputali v locen fajl z imenom nase skripte + suffixom .errors.

# Na koncu bo echojal text /var/log/messages END in nato echojal se praznino.

# Tako bomo lepo videli, kje se zacnejo entriji za dolocen log fajl in kje se

# koncajo, ter vse to loceno s presledkom pred parsi naslednjega log fajla.

# V drugem loopu argument log dobi vrednost $log2 itd in postopek se ponovi.

for log in $log{1,2,3,4,5}

do

if [ -e $log ] && [ -s $log ]

then

echo $log BEGIN

grep -E "$mydate" $log 2> $myscript.errors

echo $log END

echo

fi

done

# V zgornjem if stavku smo outputali grep v standard error fajl z imenom

# $myscript.errors. Tudi ce grep pri svojem parsanju ni dobil nikakrsnih

# errorjev, se bo ta fajl kreiral, a bo prazen. Ta spodnji if stavek poskrbi,

# da ce fajl obstaja in ce je prazen (zerobyte), potem ta fajl skripta izbrise.

if [ -e $myscript.errors ] && [ ! -s $myscript.errors ]

then

rm -rf $myscript.errors

fi

#END