Shell

Tipovi i osnovne karakteristike

Kao što smo ranije već jednom spomenuli, logovanje na sistem preko konzole nas automatski dovodi u shell. Ako sistem omogućava grafički login onda nakon logovanja pokretanje terminal programa, tj. konzole dovodi korisnika u shell. Shell je program koji prihvata i izvršava korisničke komande. Postoji više različitih shell-ova koji se razlikuju ali su u suštini vrlo slični. Iako može da deluje zbunjujuće zašto postoji više tipova shell-a to bi trebalo shvatiti kao neku vrstu evolucije. Prvi razvijen shell za Linux je bio Bourne shell, koji bi danas bio izuzetni komplikovan i nepraktičan u poređenju sa shell-ovima koji su došli kasnije. Nabrojaćemo shell-ove koji su danas prisutni u Linux sistemima, tako da korisnik može da odabere shell koji njemu najviše odgovara.

- Bash – Bourne Again Shell je trenutno najviše korišćeni shell (ujedno i najmoćniji). Kompatibilan je sa Bournovim shell-om i kreiran i distribuiran u okviru GNU projekta.
- Csh – C shell. Razvijen na Berkeley-ju. Prilično je kompatibilan sa Bournovim shellom ali ima veoma različit interfejs za programiranje. Ne dozvoljava editovanje iz komandne linije.
- Ksh – Korn shell. Možda i najpopularniji shell na Unix sistemima generalno, i prvi koji je uveo moderno i shell tehnike, od kojih je neke nasledio od C shell-a. Kompatibilan je sa Bournovim shellom i omogućava editovanje iz komandne linije.
- Sh – Bourne shell, originalni shell. Dozvoljava editovanje iz komandne linije.
- Tsch – unapređeni C shell. Omogućava editovanje iz komandne linije.
- Zsh – Z shell. Najnoviji od svih shell-ova. Kompatibilan je sa Bournovim shellom i omogućava editovanje iz komandne linije.

Ako korisnik želi da proveri koji shell koristi trenutno to može da uradi sledećom komandom:

```
$ echo $SHELL
/bin/bash
```

Najveća je verovatnoća da je pokrenut upravo bash shell jer je on ubedljivo najpopularniji Linux shell.
Elementarna upotreba shell-a je prilično nedvosmislena: ukucate komandu i shell će odgovoriti izvršavanjem te komande. Međutim, da biste izvukli maksimum iz tekstualne interakcije sa Linuxom, morate da znate i naprednije funkcije shell-a. Na kraju krajeva, većinu interaktivnih shell funkcija čine one koje vam štede vreme, a neke vam čak omogućavaju da promenite izgled shell-a i nazive njegovih komandi.

Neki od trikova koji će vam svakako uštedeti vreme:

1) PARCIJALNO KUCANJE KOMANDI

Kucanje dugačkih komandi ili imena fajlova može da uspori operacije u komandnoj liniji, produžavajući vreme kucanja, kao i povećavajući verovatnoću da se pojavi greška u kucanju, dovodeći tako do neizvršenja komande ili nekih drugih neželjenih posledica. Zbog toga bash i neki drugi shell-ovi podržavaju funkciju poznatu kao završavanje komandi. Ukucajte prvih nekoliko slova komande ili imena fajla i pritisnite taster Tab. Shell će da locira sve komande na putanju ili imena fajlova koja počinju sa unetim karakterima. Ako samo jedna komanda ili ime fajla odgovara kriterijumu, shell će da ispiše ostatak komande, odnosno ime fajla. Ako više od jedne komande ili imena fajla odgovara kriterijumu, shell će da bipne, prikaže sve komande ili imena koja dolaze u obzir, ili i jedno i drugo u zavisnosti od konfiguracije shella. Ako nema nijednog fajla koji odgovara definisanim slovima, shell će samo da bipne.

2) PONOVNO POKRETANJE KOMANDI, ISTORIJA

Dok radite u shellu postoje velike verovatnoće da ćete hteti da primenite istu komandu, ili neku njenu varijantu, ponovo i ponovo. Na primer, možda ćete probati da instalirate paket, samo da biste otkrili da pre njega morate da instalirate drugi paket. Nakon što instalirate neophodni paket, želećete da instalirate prvi. U cilju bržeg izvršenja zadatka možete da upotrebite popularnu shell funkciju: istoriju. Najmoderniji shellovi čuvaju informacije o nedavno otkucanim komandama. Možete da se krećete kroz ove komande tako što ćete pritiskati strelice tastera za gore i dole.

3) MODIFIKOVANJE KOMANDI

Shell-ovi poseduju jednostavne opcije za editovanje komandi koje kucate. Možete da upotrebite ove opcije za ispravljanje grešaka koje nastaju kucanjem komandi, ili možete da ih upotrebite za modifikovanje komandi koje ste našli pretraživanjem istorije, dodatno proširujući korisnost funkcije istorije. Najvažnije od ovih komandi su:

- Ctrl + A pomera kursor na početak linije
- Ctrl + E pomera kursor na kraj linije
- Esc + F pomera kursor za jednu reč u desno
- Esc + B pomera kursor za jednu reč u levo
- Ctrl + T menja mesto karaktera na kojem se nalazi kursor i prethodnog
- Esc + T menja mesto reči na kojoj se nalazi kursor i prethodne reči
- Ctrl + K briše od kursora do kraja linije
- Ctrl + U briše od kursora do početka linije

4) POKRETANJE VIŠE PROGRAMA: POZADINSKE OPERACIJE

Čak i u tekstualnom režimu rada, u Linuxu možete da pokrenete više programa. Jedna alatka kojom ovo možete da radite je ampersand operater (&). Dodajte ovaj karakter na kraj komande i Linux će pokrenuti program u pozadini; program će da radi, ali vi zadržavate kontrolu nad terminalom sa kojeg ste pokrenuli program. Naravno, ova akcija ima više smisla u nekim programima nego u drugim – pokretanje tekstualnog editora u pozadini baš i nema mnogo smisla, pošto morate da stupite u interakciju sa programom. Međutim, program koji samo obrađuje podatke i ne kreira nikakav konzolni rezultat, ima smisla pokrenuti u pozadini.

5) MANIPULISANJE SA ULAZOM I IZLAZOM: PREUSMERAVANJE

Linux programi koji rade u tekstualnom režimu rada rade na tri ulazno/izlazna toka, pri čemu se svaki od njih tretira kao fajl. Standardni input (stdin) je obično vezan za tastaturu i predstavlja metod pomoću kojeg program prima ulaz od korisnika. Standard output (stdout) je obično vezan za ekran, xterm prozor ili drugu alatku za prikaz u tekstualnom režimu rada i služi za prikazivanje normalnog programskog izlaza. Standard error (srderr) je takođe obično vezan za prikaz u tekstualnom režimu rada, ali je on zadužen za poruke visokog prioriteta, kao što su izveštaji o greškama. Linux može bez problema da preusmerava ove ulazne i izlazne tokove. To znači da korisnik može da uhvati izlaz programa koji radi u tekstualnom modu u fajl, ili može poslati programu sadržaj fajla u obliku ulaza. Ovo se radi koristeći operatere za preusmeravanje, koji su prikazani u tebeli koja sledi.

operator	akcija
<	preusmerava stdin da koristi specifikovani fajl
>	preusmerava stdout u specifikovani fajl, prepisujući postojeći sadržaj fajla
>>	preusmerava stdout u specifikovani fajl, pripajajući ga postojećem sadržaju fajla
2>	preusmerava stderr u specifikovani fajl, prepisujući postojeći sadržaj fajla
2>>	preusmerava stderr u specifikovani fajl, pripajajući ga postojećem sadržaju fajla
&>	preusmerava i stdout i stderr u specifikovani fajl, prepisujući postojeći sadržaj fajla

Svaki operater uzima ime fajla kao parametar. Možete da kombinujete veći broj operacija. Na primer, da biste sadržaj data.txt koristili kao ulaz za izvršnu datoteku *numcrunch*, a snimili izlaz programa u out.txt možete da ukucate:

```
$ numcrunch < data.txt >out.txt
```

6) KOMBINOVANJE KOMANDI: PIPES

Pipes nude način za povezivanje programa. Standardni izlaz prvog programa se preusmerava u drugi program, kao standardni ulaz. Lanac može da se nastavlja na onoliko programa koliko želite. Pipe operater je vertikalna crta (|), što znači da će povezana serija komandi izgledati ovako:

```
$ ps ax | grep gdm
```

Ovaj primer uzima izlaz od ps ax i „pipuje“ ga u grep, koji traga za linijama koje sadrže niz gdm. Upotrebom grepa na ovaj način možete da smanjite veličinu izlaza redundantnih programa. Možda ćete takođe želeti da „pipujete“ tekstualni izlaz kroz less pager, omogućavajući vam da pregledate izlaz brzinom koja vama odgovara.
Evo još jednog primera koji povezivanjem nekoliko naredno prikazuje najveću datoteku u trenutnom direktorijumu i pod-direktorijumima:

```
find -type f -printf '%s %p\n' |sort -nr | head -1
```

U primeru gore, find pronalazi sve datoteke (type -f), prikazuje veličinu (%s) i naziv datoteke (%p), nakon čega najpre sortira rezultate (sort) u opadajućem redosledu, a zatim prikazuje prvi (najveću datoteku) sa *head* komandom.

Shell skripte

I obični korisnici i sistem administratori mogu da koriste shell skripte za lakši rad sa Linuxom. Shell skripte vam omogućavaju da kombinujete više programa u cilju kreiranja potencijalno mnogo složenije celine. Takođe, možete da upotrebite funkcije kao što su varijable, uslovni iskazi i petlje za kreiranje složenih programa. Većina Linuxovih sistemskih skripti za startovanje su u stvari shell skripte; zbog toga, sposobnost razumevanja i modifikovanja shell skripti je od suštinske važnosti za efikasnu administraciju Linux sistema.

Shell skripte su jedna specifična podklasa interpretiranih programa, što znači da specijalni program (interpreter – u ovom slučaju shell program) čita fajl programa (shell skript) i implementira operacije koje on definiše svaki put kada se program pokrene. Ova operacija je u suprotnosti sa kompajliranim programima, u kojima kompajler čita fajl programa (izvorni kod) i generiše novi fajl (objektni kod ili binarni fajl) koji računar može da pokreće bez dalje pomoći kompajlera. Interpretirani programi, a samim tim i shell skripte, se lakše razvijaju, pošto možete da napravite izmenu u programu i odmah ga pokrenete. Interpretirani programi se takođe mogu izvršavati na bilo kom procesoru, pod uslovom da postoju odgovarajući shell i da program ne zavisi od specifičnog hardvera, ili karakteristika procesora. Za razliku od njih, kompajlirani programi su brži, pošto kompajler može da generiše objektni kod koji je optimizovan za procesor. Međutim, dobijeni kod će se moći izvršavati samo na datom procesoru ili familiji procesora.
Shell skripte obično počinju linijom koja ih identifikuje kao takve. Ova identifikaciona linija počinje karakterom #, praćenim putanjom ka interpreteru (samom shell-u). Linuxov kernel zna da interpretira ove karaktere kao identifikaciju skripte i svi jezici skriptovanja koriste kardinalni znak (#) kao karakter komentara, zato shell ignoriše ovu liniju. Mnogi shell skripte se identifikuju kao da ih pokreće /bin/sh, ali neki specifikuju /bin/bash, ili neki drugi shell. Neki jednostavni skripte mogu da se izvršavaju u širokom varijetetu

shellova, dok drugi koriste funkcije karakteristične za određene shellove. Na primer, bash i tcsh koriste različite metode za dodeljivanje vrednosti varijablama, što znači da će bilo koji skript koji ovo radi moći da se pokrene na jednom ili drugom, ali ne u oba. Da biste mogli da pokrenete shell skript, morate ili eksplicitno da ga usmerite ka shellu (kao u */bin/bash imeskripte*), ili da promenite dozvole na fajlu skripte, kako bi bio izvršni. Nakon toga možete da ga pokrenete isto onako kako pokrećete bilo koji drugi izvršni fajl, kucanjem njegovog imena (moguće sa celom putanjom ka fajlu pre njegovog imena). Da biste fajl napravili izvršnim traba da upotrebite chmod komandu:

```
$ chmod a+x imeskripte
```

Ova komanda dodeljuje svima (zbog slova a – all) pravo da izvršavaju (zbog slova x) skriptu koja se zove *imeskripte*.

Jedna od najosnovnijih upotreba shell skriptova jeste za pokretanje eksternih programa. To se radi tako što se postavlja ime eksternog programa, po mogućstvu sa kompletnom njegovom putanjom, pre imena, u skript, isto kao što biste ukucali komandu u shellovom promptu. Ako uključite ampersand (&), komanda će pokrenuti program u pozadini i procesiranje se nastavlja do sledeće komande, u suprotnom prva komanda se mora završiti pre nego što se izvrši sledeća. Takođe, možete da kombinujete programe u pipe, ili upotrebite preusmeravanje. Ako često uhvatite sebe kako kucate dugačke linije komandi, možda ima smisla da unesete tu komandu u skript i date joj kraće ime. Možete čak da upotrebite parametre poslate skriptu kao varijable, za upravljanje sa varijabilnim podacima koje komanda traži.
Mnoge linux komande su najkorisnije kada se pokreću u skriptama. Tabela koja sledi daje listu ovih komandi. Neke od ovih komandi su veoma složene, tako da bi trebalo da konsultujete stranice sa uputstvom za njih (man pages), za dodatne detalje. Naravno, možete da koristite i komande koje obično kucate u shellovom promptu, kao što su cd, cp, dd i tako dalje.

KOMANDA	EFEKAT
awk	jednostavni programski jezik koji daje rezultate na osnovu poklapanja patterna. Linux se isporučuje sa GNU awk-om ili gawk-om; on odgovara na oba imena
cut	uklanja delove liniji ulaza
echo	šalje komandu ili varijablu stdout-u
false	ne radi ništa, već vraća neuspešan kod
find	traga za fajlovima koji odgovaraju specifikovanom kriterijumu, kao što je ime fajla ili datum kreiranja.
grep	traga za fajlovima koji sadrže definisani niz. U normalnoj situaciji, šalje linije koje se poklapaju stdout-u.
pidof	vraća ID procesa (PID) koji odgovara specifikovanom aktivnom programu
read	traži ulaz od korisnika ili od fajla
sed	editor niza; edituje fajlove na osnovu komandi poslatih sed-u u komandnoj liniji, ili u skriptu
sort	sortira linije stdin-a, ili fajla i šalje sortiranu verziju stdout-u
uniq	ukljanja duplikate linija iz sortiranog fajla, ili iz stdin-a
true	ne radi ništa, već vraća uspešan kod

UPOTREBA VARIJABLI

Programi, uključujući shell skriptove, često moraju da rade sa podacima koji su nepoznati programeru. Na primer, možete da napišete skript za konvertovanje fajla sa podacima iz jednog formata u drugi, putem pozivanja serije pomoćnih programa za konverziju. Kada pišete skript, vi ne znate ime originalnog fajla, zato skript koristi jednu ili više varijabli koje će da drže ovu informaciju. Varijable okruženja su jedan tip varijabli koje možete da koristite i definišete u shell skriptama. U bash-u vi definišete varijablu sa operatorom u obliku znaka jednakosti (=), kao što je ovde prikazano:

Varijabla = vrednost

Pazite da ne bude razmaka oko operatora znaka jednakosti. Vrednost može da bude alfanumerički niz, kao hitchhiker.txt ili numerička vrednost, kao broj 42. Ako postavite znake navodnika ono tekstualnog niza, možete da uključite i razmake u vrednost varijable.
Da biste koristili varijablu, treba da upotrebite ime varijable sa znakom dolara ($) ispred nje, kao kada biste koristili konstantnu vrednost. Na primer, listing koji sledi prikazuje skript koji manipuliše sa varijablom. Ovaj skript traži i prima ulaze (koristeći echo i read komande), a onda prikazuje rezultat manipulacije. Ova manipulacija šalje varijablu komandi cut koristeći pipe, a onda ekstrahuje drugo polje (-f2), onako kako je definisano sa razmacima u ulazu (-d" "):

```
#! /bin/bash
echo "Molim vas unesite tri reči razdvojene space-om: "
read inputline
echo -n "Druga reč je: "
echo $inputline | cut -d " " -f 2
```

ako se ovaj skript zapamti kao vardemo, pa potom napravi izvršnim (sa *chmod a+x vardemo*), pokrećemo ga i dobijamo rezultat:

```
$ ./vardemo
Molim vas unesite tri reči razdvojene space-om:
jedan dva tri
Druga reč je: dva
```

Takođe, mogu se koristiti varijable $1, $2 i tako dalje, kako bi se označili parametri poslati skripti prilikom pokretanja programa. Na primer, ako ukucate

`./vardemo` *`aparam`*

$1 preuzima vrednost *aparam*. Možete da koristite ovu funkciju za lako slanje podataka skripti. Na primer, ako hoćete da napišete skript koji će da konvertuje jedan format fajla u drugi, pomoću različitih pomoćnih programa za konverziju, možete da pošaljete ime originalnog fajla kao parametar i pozivate ga kao $1 kroz ceo skript. $0 varijabla se odnosi na ime koje ste koristili za pokretanje skripte, koje može biti korisno za štampanje poruka o greškama, ili za promenu ponašanja skripte, u zavisnosti od toga kako se poziva.

UPOTREBA USLOVNIH ISKAZA

Uslovni iskaz je iskaz koji određuje da li je nešto istinito ili neistinito. Na primer, iskaz može da testira da li fajl postoji, ili da li su dve varijable jednake. Ovi iskazi se nalaze unutar pravouglih zagrada ([]) i koristi ih nekoliko komandi koje ćemo sada opisati (if, while i until). Upotreba uslovnih iskaza omogućuje shell skriptama da izvode akcije samo onda kada određeni uslov postoji, ili da izvode akcije više puta, sve dok je uslov provere zadovoljen. Tabela koja sledi daje pregled uslovnih iskaza. Pošto ovi iskazi nisu korisni osim u kombinaciji sa drugim komandama, specifični primeri njihove primene će biti prikazani u nastavku.

uslovni iskaz	značenje
-b file	istina ako fajl postoji i ako je blok fajl
-c file	istina ako fajl postoji i ako je character fajl
-d file	istina ako fajl postoji i ako je direktorijum
-e file	istina ako fajl postoji
-f file	istina ako fajl postoji i ako je običan fajl
-r file	istina ako fajl postoji i ako se može pročitati
-s file	istina ako fajl postoji i ako je nije prazan
-w file	istina ako fajl postoji i ako se može pisati u njega
-x file	istina ako fajl postoji i ako se može izvršiti
-L file	istina ako fajl postoji i ako je simbolični link
-N file	istina ako fajl postoji i ako je modifikovan od njegovog poslednjeg pristupa
file1 –nt file2	istina ako file1 ima noviji datum modifikacije od file2
file1 –ot file2	istina ako file1 ima stariji datum modifikacije od file2
-o optname	istina ako je opcija optname omogućena u shell-u
-z string	istina ako je dužina stringa nula
-n string	istina ako dužina stringa nije nula
string1=string2	istina ako su stringovi jednaki
string1!=string2	istina ako stringovi nisu jednaki
string1<string2	istina ako string1 dolazi leksikografski pre stringa2
string1>string2	istina ako string1 dolazi leksikografski posle string2
arg1 –eq arg2	istina ako je arg1 jednak sa arg2. Oba moraju biti celi brojevi.
arg1 –ne arg2	istina ako arg1 nije jednak sa arg2
arg1 –lt arg2	istina ako je arg1 manji od arg2
arg1 –gt arg2	istina ako je arg1 veći od arg2
arg1 –le arg2	istina ako je arg1 manji ili jednak sa arg2
arg1 –ge arg2	istina ako je arg1 veći ili jednak sa arg2

UPOTREBE IF I CASE

skripte često moraju da izvode različite akcije u zavisnosti od određenih uslova. Na primer, možda ćete hteti da skript prekine da se izvršava ako ne može da nađe fajl koji je korisnik specificirao, ali da nastavi sa procesiranjem ako fajl postoji; ili ćete možda želeti da prikažete informaciju koja je prilagođena za svaki od nekoliko mogućih ulaza. If i case instrukcije su zadužene za ove situacije.

PRAVLJENJE BINARNIH IZBORA: IF

If je ključna reč i koristi uslovni iskaz da bi odredila da li da izvrši ili ne određenu akciju. Njena osnovna sintaksa je :

```
if [uslov]
then
      action
else
      other action
fi
```

možete da izostavite else klauzulu i other action ako ne želite da skript radi bilo šta drugo, u slučaju da uslovi nisu ispunjeni. Za primer korišćenja ključne reči if, pogledajte listing koji sledi.

```
#!/bin/bash
read -p "Enter value of i :" i
read -p "Enter value of j :" j
read -p "Enter value of k :" k

if [ $i -gt $j ]
then
    if [ $i -gt $k ]
    then
        echo "i is the greatest"
    else
        echo "k is the greatest"
    fi
else
    if [ $j -gt $k ]
    then
        echo "j is the greatest"
    else
        echo "k is the greatest"
    fi
fi
```

Sledeća skripta omogućava kompajliranje modula drajvera i zamenu modula koji je trenutno učitan. Da bismo to uradili, moramo prvo proveriti da li je odgovarajući modul učitan ili ne. Ako jeste, uklanjamo ga iz kernela komandom rmmod, nakon čega prelazimo na kompajliranje i pravljenje nove verzije modula. Naziv modula kao i direktorijum u kojem se nalaze izvorne datoteke modula prosleđuju se skripti kao parametri prilikom poziva skripte. Na primer, ako se skripta zove rm_ins_mod pozvali bismo je sa (ako se modul zove vga):

```
pteodorovic@B150GTN:~/bash$ rm_ins_mod vga /home/user/kernel-mod/src

#!/bin/bash

if [ -c /dev/$1 ]; #proveri da li postoji device file /dev/vga
then
        rmmod $1 #ako postoji ukloni ga iz kernela
fi

tmpdir=`pwd` #zapamti trenutni direktorijum
cd $2 #promeni radni direktorijum na parametar prosleđen kroz parametar
make
insmod $1.ko #uključi novi modul u kernel
cd $tmpdir #vrati se u originalni direktorijum
```

Gornja skripta može biti pozvana iz proizvoljnog direktorijuma, obzirom da u tmpdir skladišti lokaciju prilikom poziva i da se na kraju skripte ponovo taj direktorijum postavlja za trenutni radni direktorijum.

PRAVLJENJE VIŠE IZBORA: CASE

If možete da koristite za pravljenje binarnih odluka – da preduzmete akciju ako je uslov ispunjen ili neku drugu akciju ako uslov nije ispunjen. Takođe, možete da ugnjezdite if instrukcije jedne unutar drugih – ustvari, opciona elif instukcija je kreirana za tu situaciju: koristite je posle then ali pre else, kako biste izvršili drugi test. Međutim, ponekad je potreban fleksibilniji test višestrukih uslova. Tu case stupa na scenu. Vi šaljete varijablu ovoj instrukciji, a onda specifikujete seriju akcija koje će biti preduzete u zavisnosti od vrednosti varijable. Case sintaksa izgleda ovako:

```
case variable in
        vrednost1) komande1
                ;;
        vrednost2) komande2
                ;;
        [...]
esac
```

možete da uključite proizvoljan broj vrednosti koje se mogu poklopiti sa varijablom. Shell izvršava komande koje su asocirane sa prvom vrednošću koja se poklapa sa varijablom. Vrednosti koje određujete mogu da sadrže džokere, slično džokerima u fajlovima koji se poklapaju sa njima. Jedna od najkorisnijih vrednosti jeste ona koja se sastoji samo od jednog asteriska (*). Ova vrednost se poklapa sa bilo kojom varijablom, što znači da kada je ona poslednja opcija u case instrukciji, ona radi veoma slično kao else klauzula if instrukcije. Listing koji sledi prikazuje case u akciji.

```
#/bin/bash
# Print a message about disk usage.
space_free=$( df -h | awk '/\/dev\/sda/{ print $5 }' | sort -n |
                  tail -n 1 | sed 's/\%//' )
```

```
case $space_free in
      [1-5]*)
            echo Plenty of disk space available
            ;;
      [6-7]*)
            echo There could be a problem in the near future
            ;;
      8*)
            echo Maybe we should look at clearing out old files
            ;;
      9*)
            echo We could have a serious problem on our hands soon
            ;;
      *)
            echo Something is not quite right here
            ;;
esac
```

Do kraja predavanja će biti potpuno jasno šta ovaj izraz znači, ali za sada: Najpre promenljivoj space_free dodeljujemo izlaz kompleksnog izraza. Nakon toga koristimo df alat koji prikazuje stanje na disku (diskovima):

```
pteodorovic@B150GTN:~/bash$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            7,8G     0  7,8G   0% /dev
tmpfs           1,6G  1,9M  1,6G   1% /run
/dev/nvme0n1p2  234G  113G  109G  51% /
tmpfs           7,9G  134M  7,8G   2% /dev/shm
tmpfs           5,0M  4,0K  5,0M   1% /run/lock
tmpfs           7,9G     0  7,9G   0% /sys/fs/cgroup
/dev/loop0       35M   35M     0 100% /snap/gtk-common-themes/319
/dev/loop3       15M   15M     0 100% /snap/gnome-logs/37
/dev/loop2      141M  141M     0 100% /snap/gnome-3-26-1604/70
/dev/loop1      2,4M  2,4M     0 100% /snap/gnome-calculator/180
/dev/loop5       13M   13M     0 100% /snap/gnome-characters/103
/dev/loop4       87M   87M     0 100% /snap/core/4917
/dev/loop6      3,8M  3,8M     0 100% /snap/gnome-system-monitor/51
/dev/sda1       916G  342M  869G   1% /data
/dev/nvme0n1p1  511M  6,1M  505M   2% /boot/efi
tmpfs           1,6G   16K  1,6G   1% /run/user/121
tmpfs           1,6G   56K  1,6G   1% /run/user/1000
```

Awk koji sledi filtrira iz ispisa samo osnovni disk (počinje sa /dev/sda) i za njega prikazuje procenat iskorišćenosti diska

```
pteodorovic@B150GTN:~/bash$ df -h | awk '/\/dev\/sda*/ {print $5 }'
1%
```

Preostali deo sortira izlaz u rastućem redosledu(ako ih ima više), uzima samo prvi (sa najviše prostora na disku) i uklanja %. Nakon toga se ta vrednost prosleđuje case proveri koja onda samo ispisuje poruke u zavisnosti od dostupnog disk prostora.

UPOTREBA PETLJI

Jedna od alatki koja se koristi u skriptama je petlja – način za izvršavanje istog koda ponovo i ponovo. Postoje dve široke klase petlji u shell skriptama. For petlja će izvršiti kod definisan broj puta. Ovaj broj možda nećete znati kada budete pisali skriptu, ali će biti poznat prilikom kreiranja petlje. Suprotno od ovoga, while i until petlje se izvršavaju sve dok se neki uslov ne ispuni.

FOR PETLJE
Osnovna sintaksa petlje je :

```
for variable in list
do
        commands
done
```

variable i *commands* su ono što možete da očekujete. *List* je varijabla ili izraz koji produkuje listu elemenata. *Commands* se izvršava jednom za svaki element u listi, a *variable* uzima vrednost svakog elementa iz liste, kod svakog prolaza kroz petlju. Na primer, u listingu koji sledi, svim datotekama iz direktorijumima prosleđenog kroz prvi argument prilikom poziva skripte, dodaje se sufix prosleđen kroz drugi argument neposredno pre već postojeće ekstenzije:

```
#!/bin/bash
for f in `ls $1`
do
      filename=`echo "$f" | cut -d '.' -f1`
      ext=`echo "$f" | cut -d '.' -f2`
      mv "$1/$f" "$1/${filename}_$2.$ext"
done
```

Promenljiva *filename* se postavlja tako što se ime datoteke prosleđuje programu *cut* koji koristi delimiter "." i vraća sve ono što se nalazi ispred njega. Promenljiva *ext*, na sličan način, postavlja se na ono što sledi iza delimitera. Na kraju, svaka od datoteka se preimenuje tako što se doda neposredno pre ekstenzije _ (underscore) i sufiks prosleđen kao drugi argument prilikom poziva skripte. Na primer, ako u direktorijumu *files* imamo datoteke *a.txt*, *b.txt* i *c.txt*, a pozovemo skriptu iz direktorijuma u kojem se nalazi direktorijum *files* sa:

```
pteodorovic@B150GTN:~/bash$ ./rename_files.sh files bckp
```

nakon izvršenja skripte u direktorijumu *files* ćemo imati datoteke *a_bckp.txt*, *b_bckp.txt* i *c_bckp.txt*.

WHILE I UNTIL PETLJE
While i until petlje su veoma slične, ali se razlikuju po svojim uslovima za izlazak: while petlja se izvršava sve dok je specifikovan uslov istinit, until petlja se izvršava sve dotle dok je specifikovani uslov neistinit (to jest, sve dotle dok ne

postane istinit). Iz ovoga sledi da ove dve vrste petlji zamenjuju jedna drugu, osim što morate da preokrenete prirodu uslovnih iskaza. Njihova sintaksa je takođe slična:

```
while [uslov]
do
      komande
done
```

ako se zameni while sa until dobija se until petlja. U listingu koji sledi predstavljena je varijanta već viđenog listinga. Kada se pokrene, ovaj skript stalno traži ulazno slovo. Kada ukucate prihvatljiv odgovor, on pokreće traženi program ili izlazi. Kada napustite ciljni program, videćete drugi prompt za pokretanje programa. Ovaj skript bi mogao da se napiše kako sa while tako i sa until; ali uslovni iskaz bi koristio != umesto = za određivanje trenutka kada će izaći.

```
#/bin/bash
progletter=t
until[$progletter=q]
do
echo "type the first letter of a program name: "
echo "emacs, vim or nano"
echo "(type ’q’ to exit) "
read progletter
case $progletter in
      e) emacs
            ;;
      v) vim
            ;;
      n) nano
            ;;
      *) echo "you didn’t type e, v or n; exiting.. "
esac
done
```

Alias

Korišćenjem *alias*-a moguće je napraviti prečice za komande koje često koristite. Npr:

alias logisim='java -jar /home/pteodorovic/logisim/logisim-generic-2.7.1.jar'

omogućava jednostavno pokretanje java programa logisim-generic-2.7.1.jar koji inače mora da se poziva sa java -jar /putanja/do/programa. Korišćenjem alias-a logisim ovaj program se pokreće samo kucanjem *logisim* pod pretpostavkom da je ova gore linija dodata u ~/.bashrc konfiguracionu datoteku (koja se automatski učitava svaki put kada se pokreće shell)

Još jedan interesantan primer je definisanje alias-a za komandu koja radi cd .. (prelazi u direkorijum “iznad”) više puta (npr umesto kucanja cd .. 6 puta, kucamo samo *updir* 6)

Da bi ovo bilo moguće, treba napraviti komandu updir:

```
alias updir='. ~/scripts/updir.sh'
```

Pri čemu je funkcionalnost komande definisana skriptom updir.sh (u mom slučaju ta skripta se nalazi u direktorijumu scripts unutar home direktorijuma).

Da bi sve ovo funkcionisalo kako treba, sadržaj shell skripte updir.sh je:

```
#!/bin/bash

UPDIR_LVL=$1 #parametar prosledjen skripti je broj koji nam treba

if [ $# -lt 1 ]; then #ako nije prosledjen broj kroz argument uzmi 1
 UPDIR_LVL=1
fi

UPDIR=""
for i in `seq 1 $UPDIR_LVL`; #i je ovde samo brojac 1..broj
do
 UPDIR="$UPDIR../"
done

cd $UPDIR #ako je broj 3 ovo ce biti cd ../../../ sto nam i treba
```

GREP AWK i SED

Ova tri shell programa su najmoćniji i najčešće korišćeni pa ćemo ih detaljnije analizirati.

GREP (Global Regular Expression Print)

GREP program pretražuje ulaznu datoteku u potrazi za datim stringom i ispisuje linije u kojima je pronađen dati string. Počevši od prve linije datoteke, grep kopira liniju u lokalni bafer, poredi je sa datim stringom za pretragu i ako je poređenje uspešno, ispisuje liniju na ekran. Ponavlja ovu proceduru za svaku liniju u datoteci. Važno je zapaziti da nigde tokom svoga rada grep ne čuva linije, menja linije ili pretražuje samo deo linije.
Na primer, za sadržaj datoteke *a_file*

```
boot
book
booze
machine
boots
bungie
bark
```

```
aardvark
broken$tuff
robots
```

Najjednostavnije korišćenje grep komande je

```
grep "boo" a_file
```

Kao rezultat, prikazaće se sve linije u kojima se pojavljuje traženi string:

```
boot
book
booze
boots
```

Ukoliko pretražujemo veliku datoteku, bilo bi svakako korisno da se osim linije prikaže i identifikator linije, kako bismo znali gde je pronađen traženi string. U tu svrhu koristimo dodatni parametar -n:

```
grep -n "boo" a_file
```

što da je izlaz

```
1:boot
2:book
3:booze
5:boots
```

Još jedan interesantan parametar je -v koji rezultuje printanjem negativnih rezultata. Drugim rečima, grep će prikazati sve linije koje ne sadrže traženi string, za primer od malopre:

```
grep -vn "boo" a_file
```

dobićemo izlaz

```
4:machine
6:bungie
7:bark
8:aaradvark
9:robots
```

Parametar -c zahteva od grep-a da ne prikazuje linije gde je došlo do prepoznavanja stringa, već samo da nam kaže koliko poklapanja je pronađeno:

```
grep -c "boo" a_file
4
```

Parametar -l rezultuje prikazom samo naziva datoteka u kojima je pronađen string koji se traži. Ova opcija je korisna kada se pretražuje veći broj datoteka kao na primer:

```
grep -l "boo" *
```

Još jedna opcija koja može biti korisna u slučajevima kada se ne pretražuju datoteke sa izvornim kodom je -i koja prikazuje poklapanja i sa malim i sa velikim slovima traženog stringa:

```
grep -i "BOO" a_file
```

Parametar -x pretražuje potpuna poklapanja (eXact match). Naredna komanda neće prikazati nijednu liniju jer se ni u jednoj liniji ne nalazi upravo ovaj string:

```
grep -x "boo" a_file
```

Konačno, -A dozvoljava specificiranje dodatnih linija koje će biti prikazane, osim linije u kojoj je došlo do poklapanja. Tako:

```
grep -A2 “mach” a_file
```
daje liniju u kojoj je došlo do poklapanja, kao i dve dodatne linije koje slede u datoteci:

```
machine
boots
bungie
```

GREP i regularni izrazi

Regularni izrazi predstavljaju kompaktan način da se opiše kompleksan šablon pretrage teksta. Korišćenjem grep alata, možemo koristiti regularne izraze prilikom pretrage. Postoje alati koji omogućavaju korišćenje regularnih izraza prilikom modifikacije teksta (*regexps*). Stringovi koje smo koristili do sada su zapravo veoma jednostavni regularni izrazi, ali regularni izrazi omogućavaju mnogo više. Bazične regularne izraze možemo da koristimo bez ikakvih dodatnih parametara. Na primer, da bismo pronašli sve linije koje se završavaju sa slovom e:

```
grep "e$" a_file
```

što će nam kao izlaz dati

booze
machine
bungie

Ako želite da koristite širi spektar regularnih izraza, mora se koristiti parametar -E (poznato i kao egrep komanda). Na primer, specijalni karakter ? će prepoznati 0 ili 1 pojavljivanje prethodnog karaktera:

```
grep -E "boots?" a_file
```

Što će nam dati:

```
boot
boots
```

Takođe, moguće je kombinovanje više pretraga korišćenjem specijalnog karaktera | kao u:

```
grep -E "boot|boots" a_file
```

```
boot
boots
```

Ako želimo da tražimo specijalne karaktere regularnih izraza, na primer, želimo da pronađemo pojavljivanja znaka $, to moramo onda da radimo na sledeći način:

```
grep '\$' a_file
```

broken$tuff

Takođe, može se koristiti i -F opcija koja specificira da se traže samo "obični" literali, a ne regularni izrazi.

AWK

AWK je jezik za traženje šablona i procesiranje, osmišljen od strane tri autora: Aho, Weinberger i Kernighan (otuda naziv). Veoma je složen, a ovde ćemo samo prikazati osnovnu funkcionalnost. Može biti veoma jednostavan za korišćenje i strogo je preporučen od strane zajednice jer spada među najmoćnije shell alate.
Osnove
Kao i grep, awk procesira liniju po liniju teksta iz ulazne datoteke. Može da ima opcionu BEGIN{} sekciju komandi koja će se izvršiti pre procesiranja datoteke, nakon nje sledi osnovna {} sekcija koja procesira red po red, a na kraju opet opciona END{} sekcija koja će se izvršiti nakon procesiranja cele datoteke:

```
BEGIN { .... initialization awk commands ...}
{ .... awk commands for each line of the file...}
END { .... finalization awk commands ...}
```

Za svaku liniju ulazne datoteke, proverava da li je zadat šablon za pretragu. Ako jeste, procesiraju se samo linije koje se podudaraju sa datim šablonom, a u suprotnom, procesiraju se sve linije datoteke. Šablon za pretragu, ako postoji,

može sadržati regularne izraze, kao u slučaju grep komande. Awk može da vrši i neke prilično sofisticirane matematičke operacije kao i operacije nad stringovima, a takođe podržava i asocijativne nizove (videćemo primer).

Awk svaku liniju vidi kao sekvencu polja, razdvojenih FS (Field Separator) karakterom. Podrazumevani FS karakter je whitespace karakter, tako da linija:

```
this is a line of text
```

ima 6 polja. U okviru awk komande $1 se odnosi na prvo polje, $2 na drugo itd. Cela linija je predstavljena sa $0. FS se postavlja u okviru interne varijable FS, tako da ako se postavi FS=":" onda će se : koristiti za razdvajanje polja u okviru linije, što može biti korisno u slučaju datoteka kao što su /etc/passwd i sličnih. Još jedna korisna interna varijabla je NR koja drži informaciju u broju trenutne linije u datoteci koja se procesira, kao i NF koji čuva ukupan broj polja u trenutnoj liniji.

Awk može da radi sa bilo kojom datotekom, uključujući i stdin, kada je najčešće korišćen u kombinaciji sa '|' komandom. Na primer, ako je listing trenutnog direktorijuma dobijen kao:

```
pteodorovic@B150GTN:~/cpp/main_libTest$ ls -l
total 40
-rw-r--r-- 1 pteodorovic pteodorovic   255 Nov 28 12:17 main.cpp
-rwxr-xr-x 1 pteodorovic pteodorovic 13616 Nov 28 12:26 main_libTest
-rw-r--r-- 1 pteodorovic pteodorovic   679 Nov 28 12:26 main_libTest.pro
-rw-r--r-- 1 pteodorovic pteodorovic  5640 Nov 28 12:26 main.o
-rw-r--r-- 1 pteodorovic pteodorovic  6605 Nov 28 12:26 Makefile
```

možemo primetiti da je veličina datoteke data u koloni 5. Tada, ako želimo da saznamo ukupnu veličinu svih datoteka unutar datog direktorijuma, to možemo dobiti sa komandom:

```
pteodorovic@B150GTN:~/cpp/main_libTest$ ls -l | awk 'BEGIN {sum=0}
{sum=sum+$5} END {print sum}'
26795
```

Jedina komanda u okviru END bloka prikazuje dobijenu sumu. Slično, trivijalnom modifikacijom bi se mogla računati srednja vrednost i standardna devijacija datih brojeva: akumuliranjem sum_x i sum_x2 u glavnom awk bloku i računanjem srednje vrednosti i standardne devijacije korišćenjem odgovarajućih formula u END bloku.

Awk obezbeđuje podršku za petlje(for i while) kao i za grananje (if). Dakle, ako želimo da preskočimo određene podatke i radimo npr samo sa svakim trećim redom u datoteci, to ćemo uraditi sa:

```
pteodorovic@B150GTN:~/cpp/main_libTest$ ls -l | awk '{for (i=1;i<3;i++)
{getline}; print NR,$0}'
3 -rwxr-xr-x 1 pteodorovic pteodorovic 13616 Nov 28 12:26 main_libTest
6 -rw-r--r-- 1 pteodorovic pteodorovic  6605 Nov 28 12:26 Makefile
```

gde for petlja koristi getline funkciju kako bi se pomerala kroz datoteku, nakon čega bi prikazala svaku treću liniju iz datoteke.

Ako se koristi šablon za pretragu, on se stavlja na prvo mesto, a tek nakon njega sledi akcija. Jedan deo od ova dva mora biti prisutan-ako nedostaje šablon, akcija će biti sprovedena na svakoj liniji datoteke, a ako nedostaje akcija, prikazaće se sve linije koje zadovoljavaju dati šablon.

U slučaju da se ne koristi šablon, posmatramo datoteku sa logovima pristupa određenim ip adresama u formatu datum vreme ip-adresa:

```
pteodorovic@B150GTN:~/iplogtest$ cat Iplogs.txt
180607 093423     123.12.23.122 133
180607 121234     125.25.45.221 153
190607 084849   202.178.23.4 44
190607 084859   164.78.22.64 12
200607 012312     202.188.3.2 13
210607 084849   202.178.23.4 34
210607 121435     202.178.23.4 32
210607 132423     202.188.3.2 167
```

Sledeća awk komanda će nam omogućiti da dobijemo broj pristupa svakoj od ip adresa:

```
pteodorovic@B150GTN:~/iplogtest$  awk '{Ip[$3]++;} END{for(var in Ip)
print var, "accessed", Ip[var]," times"}' Iplogs.txt
125.25.45.221 access 1  times
123.12.23.122 access 1  times
164.78.22.64 access 1  times
202.188.3.2 access 2  times
202.178.23.4 access 3  times
```

U gornjem primeru vidimo i upotrebu asocijativnih nizova u okviru awk komande, gde se umesto indeksiranja niza koriste stringovi koji predstavljaju ip adrese iz log datoteke.

Za primer korišćenja šablona, posmatramo sadržaj direktorijuma

```
pteodorovic@B150GTN:~/cpp/main_libTest$ ls -l | awk '{print NR,$0}'
1 total 44
2 -rw-r--r-- 1 pteodorovic pteodorovic   255 Nov 28 12:17 main.cpp
3 -rwxr-xr-x 1 pteodorovic pteodorovic 13616 Nov 28 12:26 main_libTest
4 -rw-r--r-- 1 pteodorovic pteodorovic   679 Nov 28 12:26
main_libTest.pro
5 -rw-r--r-- 1 pteodorovic pteodorovic  5640 Nov 28 12:26 main.o
```

```
6 -rw-r--r-- 1 pteodorovic pteodorovic  6605 Nov 28 12:26 Makefile
7 drwxr-xr-x 2 pteodorovic pteodorovic  4096 Dec 26 09:31 testDir
```

Ako želimo da prikažemo samo datoteke, bez direktorijuma (u ovom slučaju testDir jedini) dodajemo izrazu šablon. Šablon se postavlja unutar // kao u primeru ispod:

```
pteodorovic@B150GTN:~/cpp/main_libTest$ ls -l | awk '/^[^d]/ {print NR,
$0}'
1 total 44
2 -rw-r--r-- 1 pteodorovic pteodorovic   255 Nov 28 12:17 main.cpp
3 -rwxr-xr-x 1 pteodorovic pteodorovic 13616 Nov 28 12:26 main_libTest
4 -rw-r--r-- 1 pteodorovic pteodorovic   679 Nov 28 12:26
main_libTest.pro
5 -rw-r--r-- 1 pteodorovic pteodorovic  5640 Nov 28 12:26 main.o
6 -rw-r--r-- 1 pteodorovic pteodorovic  6605 Nov 28 12:26 Makefile
```

Za kontrolu toka izvršavanja u awk komandama, koriste se:

```
if (condition) statement [ else statement ]
while (condition) statement
do statement while (condition)
for (expr1; expr2; expr3) statement
for (var in array) statement
break
continue
exit [ expression ]
```

Za ulazno/izlazne operacije koriste se komande:

getline	postavi $0 iz naredne linije ulaza
getline < file	postavi $0 iz naredne linije ulaza datoteke
getline var	postavi promenljivu var iz naredne linije ulaza
getline var < file	postavi promenljivu var iz naredne linije ulaza date datoteke
next	Stani sa procesiranjem trenutne linije. Naredna linija je pročitana i procesiranje počinje sa prvim prepoznatim šablonom ako je dat. Ako je bila poslednja linija, END blok, ukoliko postoji će biti izvršen
nextfile	Stani sa procesiranjem trenutne datoteke. END blok, ako postoji, će biti izvršen
print	Prikaži trenutnu liniju
print expr-list	Prikaži listu iskaza
print expr-list < file	Prikaži listu iskaza iz datoteke
printf fmt,exp-list	Formatiraj i prikaži (koristeći C sintaksu)

Awk numeričke funkcije uključuju:

atan2(y, x)	Arkus tangens y/x radijana

cos (expr)	Kosinus izraza u radijanima
exp(expr)	Eksponencijalna funkcija
int(expr)	Zaokružuje na celobrojni podatak
log(expr)	Prirodni logaritam
Rand()	Vraća slučajan broj N (između 0 i 1)
sin(expr)	Sinus izraza
sqrt(expr)	Kvadratni koren izraza
srand([expr])	Koristi iskaze kao seed za generator slučajnih brojeva, ako se ne prosledi iskaz, koristi se trenutno vreme dana

Awk funkcije za rad sa stringovima

gsub(r, s, [, t])	Za svako poklapanje dela stringa u stringu t sa regularnim izrazom r, zameni na odgovarajuće mesto string s i vrati broj zamena, ako se ne prosledi t, koristiće se $0. Npr echo "Testiramo awk" | awk 'gsub(/awk/,"vim")'
index(s, t)	Vraća indeks stringa t u stringu s, ili 0 ako ne postoji
length([s])	Vraća dužinu stringa s, ili dužinu $0 ako nije prosleđen s
match(s, r, [, a])	Vraća poziciju unutar s na kojoj je pronađen regularni izraz r ili 0 ako ne postoji
split(s, a, [, r])	Deli string s u niz a, koristeći regularni izraz r, a vraća broj elemenata niza. Ako r nije prosleđen, koristi se FS
sprintf(fmr, expr-list)	Prikazuje expr-list u skladu sa formatom fmt i vraća rezultat
strtonum(str)	Ispituje str i vraća numeričku vrednost
sub(r, s, [, t])	Isto kao gsub(), ali samo prvo poklapanje se zamenjuje
substr(s, i [, n])	Vraća najviše n karaktera pod-stringa s počevši na poziciji i. Ako n nije prosleđeno, ostatak stringa se vraća
tolower(str)	Vraća kopiju stringa str gde su sva mala slova promenjena u velika
toupper(str)	Vraća kopiju stringa str u kome su sva velika slova promenjena u mala

Korišćenjem parametra -v može se awk programu proslediti promenljiva. Npr

```
awk -v skip=3 '{for (i=1;i<skip;i++) {getline}; print $0}' a_file
```

Takođe, moguće je napisati awk program i sačuvati ga kao posebnu skriptu, što je veoma zgodno u slučajevima kompleksnijih awk izraza:

```
#!/usr/bin/awk -f
#only print out every 3rd line of input file
BEGIN {skip=3}
```

```
{for (i=1;i<skip;i++)
{getline};
print $0}
```

Naravno, ako se ova skripta sačuva pod nazivom print_nth_line, tada joj se moraju najpre dodeliti prava da bude izvršna skripta komandom

```
chmod u+x print_nth_line
```

nakon čega se poziva sa

```
./print_nth_line my_file.txt
```

SED (Stream EDitor)

sed izvršava bazične transformacije na ulaznom toku (datoteci ili ulazu iz cevi-pipe) tokom jednostrukog prolaska kroz tok, tako da je veoma efikasan.

Sed se može koristiti iz komandne linije, ili u okviru skripte, kako bi se editovao sadržaj datoteke bez izmene same datoteke.
Možda najkorisnija primena sed alata je za pronalaženje i zamenu "search-and-replace" jednog stringa u drugi.

Može se koristiti u sklopu komandne linije sa parametrom -e, ili se može upisati cela sed komanda u zasebnu datoteku (npr sed.in), koja će onda biti pozvana od strane sed programa sa opcijom '-f sed.in'. Ovakvo korišćenje je zgodno kada je sed komanda kompleksna i sadrži mnogo regularnih izraza.
Komandom

```
sed -e 's/input/output/' my_file
```

će prikazati svaku liniju datoteke my_file, zamenjujući prvo pojavljivanje stringa "input" na svakoj liniji u "output".
Sed je takođe linijski orijentisan, tako da ukoliko želimo menjati sva pojavljivanja u svakoj od linija to radimo sa:

```
sed -e 's/input/output/g' my_file
```

Izrazi između /.../ mogu biti stringovi literala ili regularni izraz.
Podrazumevano, izlaz se šalje na stdout, ali može biti preusmeren u drugu datoteku ili ukoliko želimo da izmene vršimo direktno na datoteci koju pretražujemo, to radimo sa'-i' parametrom:

```
sed -e 's/input/output/' my_file > new_file
sed -i -e 's/input/output/' my_file
```

Šta ako je neki od karaktera izraza koji koristimo za pretragu specijalan simbol, kao na primer '/'? Tada se taj karakter mora koristiti sa \ uspred njega. Na primer, ukoliko želimo da editujemo shell skriptu kako bismo sve reference /bin promenimo na putanju /usr/local/bin , to ćemo uraditi ovako

```
sed -e 's/\/bin/\/usr\/local\/bin/' my_script > new_script
```

Nekada ovakav zapis može predstavljati problem, pa se tada može zameniti podrazumeveno značenje / karaktera na sledeći način:

```
sed -e 's,/bin,/usr/local/bin,' my_script > new_script
```

Sada , postaje delimiter koji se koristi da odvoji šablon za pretragu od stringa koji će se koristiti umesto njega.
Specijalni karakter '&' pokazuje na prepoznati šablon. Na primer, ako imamo datoteku u kojoj svaka linija počinje brojem i želimo da taj broj stavimo u zagradu, to ćemo uraditi sa

```
sed -e 's/[0-9]*/(&)/' my_file
```

gde je [0-9] regularni izraz koji se odnosi na cifre a * označava ponavljanje prethodnog karaktera, što znači proizvoljan broj cifara, koliko god da ih ima.

Dodatne sed komande

Generalna forma je

```
sed -e '/pattern/ command' my_file
```

gde je pattern regularni izraz a command može biti 's' za search and replace, 'p' za print, 'd' za delete, 'i' za insert, 'a' za append itd. Podrazumevano ponašanje je da se prikažu i linije kod kojih nije došlo do poklapanja sa šablonom pretrage, tako da, ukoliko želimo da njih ne prikazujemo, koristimo -n parametar, a onda sa p komandom kontrolišemo šta će biti prikazano ispisom. Na primer, ako želimo listing svih poddirektorijuma (bez datoteka u datom direktorijumu) možemo da koristimo

```
ls -l | sed -n -e '/^d/ p'
```

pošto ls -l počinje sa karakterom d za svaki direktorijum (videli smo već ranije na primeru awk-a), prethodna komanda će ispisati samo takve linije koje počinju 'd' simbolom.
Slično, ako želimo da obrišemo sve linije koje počinju sa simbolom # (obrišemo komentare), to ćemo uraditi sa

```
sed -e '/^#/ d' my_file
```

Takođe, može da se koristi i opseg na kome se izvršava komanda:

```
sed -e '1,100 command' my_file
```

će izvršiti komandu 'command' na linijama 1 do 100. Ovde se može koristiti i karakter $ da označi kraj datoteke. Tako, ako želimo da obrišemo sve osim prvih 10 redova date datoteke, to ćemo uraditi sa

```
sed -e '11,$ d' my_file
```

Evo još jednog primera u kojem se sed koristi da pronađe XML datoteku sa traženim XML elementom <Name> i prikazuje vrednost dodeljenu tom XML elementu:

```
find . -name '*.xml' -exec sed -n 's/<Name>\([^<]*\)<\/Name>/\1/p' {} +
```

Na primer, za user.xml datoteku koja izgleda ovako

```
<user>
        <Name>Predrag Teodorovic</Name>
        <Office>402</Office>
        <Role>Teacher</Role>
</user>
```

Ukoliko se datoteka user.xml nalazi u trenutnom direktorijumu ili bilo kojem pod-direktorijumu trenutnog direktorijuma, gornja naredba će ispisati vrednost prepoznatog XML tag-a :

```
pteodorovic@B150GTN:~/bash$ find . -name '*.xml' -exec sed -n 's/<Name>\
([^<]*\)<\/Name>/\1/p' {} +
      Predrag Teodorovic
```

Primer gore pronalazi datoteke sa ekstenzijom .xml, a zatim koristi sed kako bi prikazao rezultat u željenom formatu. {} će se zameniti sa pronađenim datotekama a + ili \; je neophodno da bi se terminirala find exec komanda.