Izvorni znanstveni članak

Značaj skriptnih jezika za administraciju operacijskih sustava

Vladimir Mateljan
Odsjek za informacijske znanosti, Filozofski fakultet
Ivana Lučića 3, Zagreb, Hrvatska
vladimir.mateljan@zg.t-com.hr

Željka Požgaj
Ekonomski fakultet
Trg. J. F. Kennedyja 6, Zagreb, Hrvatska
zpozgaj@efzg.hr

Krunoslav Peter
Filozofski fakultet
Ivana Lučića 3, Zagreb, Hrvatska
kruno_peter@yahoo.com

Sažetak

Skriptni programski jezici, poput jezika `bash` *i* `awk`*, omogućuju rapidni razvoj programskih rješenja koja se koriste za automatizaciju poslova administracije operacijskih sustava, uz minimun uporabljenog programskog koda. Ovaj članak objašnjava osnovni razlog za pragmatičnost skriptnih jezika – njihov dizajn primjeren je za izradu miminalističkih programskih rješenja u formi naloga za naredbeni redak, aliasa i skripti na način da se ostvaruje integracija gotovih komponenti – naredbi i programskih rješenja, iz razvojnog okruženja – ljuske operacijskog sustava. Članak identificira neke dodatne beneficije koje proizlaze iz primjene skriptnih jezika u poslovima administracije operacijskih sustava.*

Ključne riječi: skriptni programski jezik, administracija, operacijski sustav, redak za upis naredbi

1. Uvod

Suvremeni trend u administriranju operacijskih sustava jest da se poslovi administracije operacijskog sustava izvode uz primjenu programskih pomagala iz slikovnog sučelja operacijskog sustava, primjeri kojih su Webmin za Unix i GNU/Linux operacijske sustave ili Microsoft Management Console (MMC) za Microsoft Windows operacijske sustave. Druga mogućnost svodi se na obavljanje poslova administracije u okruženju ljuske operacijskog sustava na način da

se, uz primjenu *skriptnih programskih jezika*, kreiraju i izvršavaju *nalozi* u retku za upis naredbi, *aliasi* i *skripte*. U članku će primjeri takvih programskih rješenja biti prikazani na primjeru skriptnih programskih jezika koji potječu iz okruženja operacijskog sustava Unix:

1. jezik ljuske `bash`.
2. `awk`, programski jezik za pretraživanje i obradu teksta,

Koncepti iz programskog jezika ljuske `bash` implementirani su u ljuskama operacijskih sustava Windows i Mac OS X. `awk` je također je ostvaren za spomenute operacijske sustave.

Drugo poglavlje članka posvećeno je definiranju termina "*skriptni jezik*". Skriptni programski jezici predstavljaju posebnu vrstu programskih jezika; prema svojoj svrsi, osobinama i pragmatici, razlikuju se od konvencionalnih programskih jezika, no s obje vrste jezika, u današnje vrijeme, mogu se ostvariti programska rješenja iste funkcionalnosti. Treće poglavlje donosi sažeti opis dva predstavnika skriptnih jezika, `bash` i `awk`. U opisima spomenutih programskih jezika nastoje se naglasiti karakteristike njihovog dizajna, koje pogoduju izradi programskih rješenja u minimalnom vremenu i uz minimalnu količinu koda.

U promišljanju značaja skriptnih programskih jezika za administraciju operacijskih sustava, polazimo od pregleda proizvoda rudimentarnog programiranja u okruženju ljuske (engl. shell) operacijskog sustava: nalog u naredbenom retku, alias i skripta. Važno obilježje ovih programskih rješenja jest jednostavnost, jer ih u praksi izrađuju administratori i korisnici operacijskih sustava kojima programiranje nije prioritetni posao.

O pragmatici i značaju skriptnih programskih jezika za administraciju operacijskih sustava progovara se u četvrtom i petom poglavlju članka. Primjenom skriptnih jezika za ostvarivanje programskih rješenja možemo automatizirati poslove administracije operacijskih sustava.

Izvorni kod naloga, aliasa i skripti pisan je vrstom slova `Courier`. Nalozi za upis u redak za naredbe pisani su tako da im je početak postavljen znak "**$**" koji se ne upisuje, jer predstavlja znak spremnosti (engl. prompt). Neki primjeri u `bash`-u lako se mogu adaptirati za izvršavanje u okruženju ljuske operacijskog sustava Windows (Torres, 2004, 2), npr. naredba `ls` iz Unixa može se zamijeniti naredbom `dir` Windowsu.

2. Skriptni programski jezici

2.1 Svrha i namjena skriptnih programskih jezika

Skriptni programski jezici su interpretirani[1] programski jezici (Ousterhout, 1997), koji su namijenjeni za:

- izradu aplikacija od gotovih komponenti,

[1] Potrebno je napomenuti da se interpretirani programski jezici mogu ostvariti kao prevodioci. Tako je za programski jezik `awk` ostvaren prevodilac s nazivom `tawk`.

- upravljanje aplikacijama koje imaju programabilno sučelje,
- pisanje programa kojih je brzina razvoja važnija od učinkovitosti u izvršavanju (Barron, 2000, 4).

U skriptne programske jezike se ubrajaju jezici ljuski, awk, perl, Python, PHP, Tcl, JavaScript, etc. Većina skriptnih jezika se temelji na paradigmi strukturnog programiranja (bash, awk, etc.); Python je primjer objektno orijentiranog skriptnog programskog jezika.

Skriptni programski jezici obično implementiraju se kao interpretatori (Barron, 2000, 6), izvodeći naredbu iz skripte odmah nakon čitanja, premda se mogu ostvariti kao prevodioci ili prevodioci u trenutku izvršavanja (engl. just in time compiler).

Postoje razne vrste skriptnih jezika. Mogu biti implemetirani samostalno ili u okruženju editora, tabličnih kalkulatora, tekst procesora, etc. U kontekstu ovog članka, pažnja će biti usmjerena na skriptne jezike za kontrolu poslova u operacijskom sustavu (engl. job control languages), u koje se ubraja `bash`, te na skriptne jezike za procesiranje teksta, iz kojih je izdvojen `awk`.

2.2 Bash i awk – predstavnici skriptnih programskih jezika

Cjeloviti opisi jezika `awk` i `bash` mogu biti preobimni čak i za jednu knjigu. U navođenju sintakse naloga oba programska jezika, opcionalni dijelovi naloga pisat će se unutar uglatih zagrada ("`[`" i "`]`"); tri točke ("`...`") označavat će ponavaljanje naloga, koji im prethodi. Naglasak u sažetom opisu `bash`-a bit će na sintaksi naredbi, ulančavanja i preusmjeravanja. U svezi s `awk`-om će se opisati: (1) sintaksa naloga za poziv `awk`-skripte u retku za opis naredbi, (2) sintaksa naloga `awk`-skripte, sastavljena od uzorka i procedure te njegova semantika.

O pragmatici ovih jezika, kao predstavnika skriptnih jezika, raspravlja se jednim dijelom u ovom poglavlju i sljedećim poglavljima članka, uz nastojanje da se naglase svojstva njihovog dizajna koja podržavaju rapidni razvoj programskih rješenja za administraciju sustava uz minimum koda.

Osnovna vrsta podataka nad kojom `bash` i `awk` izvode operacije jest *znakovni niz* (engl. string), odnosno *tekst*. Takav dizajn skriptnih jezika omogućuje implementaciju ulančavanja i preusmjeravanja *tekstovnog tijeka* (engl. text stream) što ćemo promotriti u opisima `bash`-a i `awk`-a.

2.3 Programski jezik ljuske bash

`Bash` je "sustavski interpretator naredbi" (Peek et al, 1998, 82); posjeduje "napredne programske sposobnosti bilo kojeg interpretatora svoje vrste" (Newham, 2005, 81). Naredbe se, u okruženju `bash`-ljuske operacijskog sustava Unix, tipkaju u retku za upis naredbi ili se nalaze u skripti; pojedinačna naredba ima jednostavnu sintaksu (Peek et al, 1998, 9):

```
naredba [prekidač...] [datoteka...]
```

U retku za upis naredbi, naredbe se mogu nizati tako da se prethodna razvoji od sljedeće znakom “**;**” (točka-zarez, engl. semicolon) (Kernighan & Pike, 1984, 71). Na taj način ostvaruje se *nalog*. Sintaksa nizanja naredbi:

```
naredba [; naredba...]
```

Ulančavanje je niz od dvije ili više naredbi povezanih znakom “**|**”:

```
naredba [| naredba...]
```

Semantika sintakse za ulančavanje je jednostavna: podaci sa standardnog izlaza prve naredbe predaju se na standardni ulaz prve naredbe. U okruženju ljuske je uobičajeno da naredbe čitaju podatke sa standardnog ulaza – tipkovnice, a izlazne rezultate prikazuju na standardnom izlazu – zaslonu (Peek et al, 1998, 67); ulančavanjem se izlazni rezultat jedne naredbe može predati drugoj u tekstovnom tijeku.
Sintaksa za ulančavanje naredbi je jednostavna i efikasna – uz minimum koda izražava se poziv druge naredbe i predaju joj se ulazni podaci.
Iz opisa sintakse `bash`-a će biti izdvojena i pojednostavljena sintaksa preusmjeravanja, još jednog koncepta, koji pridonosi rapidnoj izradi programskih rješenja uz minimum koda – izlazni podaci iz naredbe `naredba` pohranjuju se u datoteci `datoteka`:

```
naredba > datoteka
```

Dodavanje sadržaja na kraj postojeće datoteke ostvaruje se primjenom simbola “**>>**”. Ovakva sintaksa, premda jednostavna, ima slojevitu semantiku: upućuje na implemetaciju algoritma za otvaranje datoteke, pohranu znakovnog niza primljenog sa standardnog ulaza i zatvaranje datoteke. Program u konvencionalnom programskom jeziku, kao što npr. C ili Pascal, s ovakvom funkcionalnošću će imati više (desetaka) redaka.
Na primjeru sintakse i semantike ulančavanja i preusmjeravanja dokazuje se tvrdnja da dizajn programskih jezika utječe na njegovu primjenjivost u rapidnom razvoju programskih rješenja i to uz ostvarivanje principa minimuma napisanog koda i maksimalne ekspresivnosti.
U Primjeru 1 prikazana je skripta u kojoj je primjenjeno ulančavanje i preusmjeravanje. Naredba `echo` prikazuje specificirani niz znakova na standardnom izlazu – zaslonu. Znakom “#” ispred retka označavaju se napomene u skripti.

Primjer 1. Popis datoteka s nizom znakova "txt" u nazivu

```
#!/bin/bash
# popis datoteka s nizom znakova "txt" u nazivu
echo "TEKSTOVI" > doc.txt
ls | grep txt >> doc.txt
```

Prvi nalog u skripti (`#!/bin/bash`) daje informaciju o interpretatoru skripte, programu koji izvodi skriptu, a to je ljuska `bash`.
Dodatakom naredbi za grananje i petlju te mogućnosti za rad s varijablama sustava (Newham, 2005, 86), programski jezik ljuske `bash` postaje u potpunosti osposobljen izradu programskih rješenja u okviru paradigme strukturnog programiranja. O pragmatici `bash`-a se detaljnije progovara u ostallim poglavljima članka.

2.4 Programski jezik awk

`Awk` je programski jezik za "pretraživanje i obradu teksta" (Aho et al, 1978). Jezik je nastao 1977. godine u Bell Laboratories. Tvorci `awk`-a su Alfred V. Aho, Brian W. Kernighan i Peter J. Weinberger; naziv jezika je kratica za prezimena autora jezika.
Osnovna svrha `awk`-a jest da pretražuje retke ulazne tekst datoteke koji sadrže određene uzorke; nakon što pronađe redak s odgovarajućim uzorkom, `awk` izvodi specificirane radnje (procedure) nad tim retkom (Robbins, 2001, 3). Uzorak može tvoriti proizvoljna Booleova kombinacija regularnih izraza (engl. regular expression) i relacijskih operatora na znakovnim nizovima (engl. strings), brojevima, poljima sloga, varijablama, i elementima asocijativnog jednodimenzionalnog polja (engl. array) (Aho et al, 1978). Procedure mogu također uključivati izraze za prepoznavanje uzoraka, aritmetičke operacije, operacije nad znakovnim nizovima, pohranu i manipulaciju podacima u asocijativnom polju, grananje, `while`- i `for`-petlju te mnostruke izlazne tijekove (Aho et al, 1978).
U Primjeru 2 prikazan je `awk` nalog za upis u naredbenom retku koji prikazuje redni broj ispred naziva datoteke. `Print` je naredba awk-a koja na standardni zaslon ispisuje sadržaj varijable ili niz znakova.

Primjer 2. Awk nalog – prikazuje redni broj ispred naziva datoteke

```
$ ls | awk '{ print NR, $0 }'
```

Skripta se u `awk`-u može, kao što je to vidljivo u Primjeru 2, zadati u retku za upis naredbi.
Sintaksa za poziv `awk`-a sa skriptom zadanom u naredbenom retku je (Robbins, 2000, 24):

```
awk [opcija] 'skripta' ulazna_datoteka
```

Skripta iz datoteke se poziva primjenom prekidača `-f`:

```
awk [opcija] -f skripta ulazna_datoteka
```

Zadavanje skripte `awk`-interpretatoru u retku za upis naredbi predstavlja mogućnost vrijednu naglaska. Na taj način mogu se u vrlo kratkom vremenu, možda čak i u sekundama, izraditi funkcionalna programska rješenja za obradu tekstovnih datoteka (ili tekstovnog tijeka (engl. text stream)). To je razlog da se `awk` ponekad naziva "alatom", premda je on programski jezik.
Sintaksu je `awk` preuzeo od programskog jezika C (Kernighan & Richie, 1988), ali je ona za `awk` u velikoj mjeri pojednostavljena. Nalog se u C-u obavezno treba završiti znakom "`;`" (Kernighan & Richie, 1988), a u `awk`-u je taj znak opcionalan. To pridonosi lakoći i brzini pisanja izvornog koda u `awk`-u.
U odnosu na C ili Pascal, kao predstavnike konvencionalnih programskih jezika, `awk` poznaje samo jednu vrstu podataka – *znakovni niz* (engl. string). Interno se obavlja konverzija stringa u broj, ili konverzija broja u string, kada to kontekst zahtijeva (Aho et al, 1978). U nastavku članka ćemo analizirati beneficije od unifikacije različitih vrsta podataka u znakovni niz.
Osobitost `awk`-a također su ugrađene (engl. built-in) varijable (Aho et al, 1978). Primjer takve varijable je `NR` u Primjeru 2, u koju `awk` automatski pohranjuje broj trenutno učitanog sloga. Ovo je osobina koja pridonosi primjerenosti `awk`-a za rapidni razvoj, uz minimum napisanog koda, jer programera oslobađa potrebe za pisanjem naredbi koda za brojanje učitanih slogova ili drugih pokazatelja.
`Awk` skripta je sastavljena od *naloga* ili *pravila* čija je sintaksa `uzorak { procedura }` (Robbins, 2000, 25). `awk` program je, dakle, niz uzoraka i akcija, koje određuju, što potražiti u ulaznim podacima i što napraviti s pronađenim podacima (Aho et al, 1988). Prema tome, struktura `awk` programa izgleda ovako (Robbins, 2001, 3):

```
uzorak { procedura }
uzorak { procedura }
...
```

Ako je u nalogu izostavljen uzorak, procedura se primjenjuje na sve učitane slogove; ako nedostaje procedura, ispisuje se slog u kojem je pronađen uzorak (Robbins, 2000, 26). Posebni uzorak, `BEGIN`, služi za izvršavanje procedure prije učitavanja prvog zapisa iz ulazne datoteke, koja služi za inicijalizaciju programa (npr. globalnih varijabli). Uzorak `END` izvršava proceduru nakon zadnjeg učitanog sloga zadanih ulaznih datoteka (Robbins, 2000, 27).

Primjer 3. Awk nalog u retku za upis naredbi, koji broji retke u kojima je pronađen uzorak "Linux" ili "linux" i ispisuje broj (frekvenciju) pojavljivanja

```
$ awk '/[Ll]inux/ { i++ } END { print i }' file.txt
```

U Primjeru 3 je prikazana primjena uzorka s regularnim izrazom `[Ll]inux` i uzorka `END` koji se izvršava nakon što se učitaju svi zapisi ulazne datoteke `file.txt`.

U programski jezik `awk`, kao i u druge alate u okruženju ljuske operacijskog sustava Unix, ugrađen je jezik za *regularne izraze*, koji `awk`-u daje izuzetne sposobnosti za pretraživanje teksta i oslobađa programera potrebe da te mogućnosti kodira: znakovni nizovi se mogu pretraživati uz specificiranje uzorka, umjesto prema fiksnim podnizovima znakovnog niza (Robbins, 2000, 2).

Do posebnog izražaja `awk` dolazi u obradi tekst datoteka s podacima, točnije rečeno, sekvencijalnih datoteka koje imaju slogove i polja unutar slogova (Robbins, 2000, 23). `awk` parsira takve ulazne datoteke i automatski dijeli svaki ulazni zapis u polja (Aho et al, 1988). "Budući da su mnoge stvari automatske – ulaz (tj. čitanje ulaznih datoteka), dijeljenje na polja, upravljanje memorijom, inicijalizacija – `awk` programi su obično manji od onih u konvencionalnim jezicima" (Aho et al, 1988). To `awk` čini kvalitetnim skriptnim jezikom i svrstava ga u red jezika, koji su primjereni za rapidni razvoj softvera. On omogućuje sustavskim administratorima da na jednostavan način ostvaruju programska rješenja, i to uz minimun koda (slobodno se može primjetiti da se `awk`-u ne pišu naredbe za otvaranje i zatvaranje ulazne ili izlazne datoteke, niti se kodira algoritam za parsiranje podataka ulazne datoteke).

`Awk` obrađuje ulazne retke, dok ne dosegne kraj ulaznih datoteka (Robbins, 2001, 3) – u programskom jeziku `awk` implementiran je algoritam koji:

- otvara ulaznu datoteku,
- parsira znakove ulazne datoteke (do trenutka nailaska na znak za kraj datoteke),
- zatvara ulaznu datoteku.

Programski jezik `awk` je specijaliziran za pretraživanje i obradu teksta, za razliku od jezika ljuske `bash`, koji je namijenjen za programska rješenja, koja automatiziraju sustavsku administraciju. `awk` se smatra programskim pomagalom za opću namjenu te može nadomjestiti ostale specijalizirane alate i programe (npr. `grep`, `cut`, `tail`, etc.). Često se primjenjuje u je izradi izvještaja (engl. report), transformaciju podataka, ili čak za generiranje izvornog koda (engl. code generator) u drugim programskim jezicima.

3. Nalog, alias, skripta

Nalog uključuje jednu ili više naredbi operacijskog sustava. U Primjeru 4 naveden je nalog koji prekida izvršavanje svakog procesa u čijem je nazivu niz znakova "sleep"; pretpostavka je da naredba `ps` daje podatak o ID-u procesa u prvom, a nazivu procesa u četvrtom stupcu.

Primjer 4. Nalog za prekid izvršavanja procesa koji u nazivu ima niz znakova "sleep"

```
$ ps | awk '$4 ~ /sleep/ { cmd = "kill -9 " $1; print | cmd }'
```

U prikazanom primjeru ostvareno je ulančavanje naredbe `ps` (prikazuje trenutno aktivne procese) i `awk`-skripte koja generira nalog s naredbom `kill` (prekida izvršavanje procesa).
Ukoliko u poslovima administriranja često ponavljamo određeni nalog, možemo ga pohraniti u obliku aliasa.

Primjer 5. Alias za broj datoteka u imeniku

```
$ alias lsx='ls | wc -l'
```

U Primjeru 5 ulančavanjem su povezane naredba `ls` (pokazuje sadržaj imenika) i `wc -l` (broji redove, riječi i znakove).
Niz naredbi možemo pohraniti u u tekst datoteku – *skriptu* (engl. script). Jednostavna skripta je dana u Primjeru 1.
Naziv "skripta" potječe iz ranih 70-ih godina (Barron, 2000, 4); tvorci operacijskog sustava Unix taj su izraz upotrebljavali za "niz naredbi koje se čitaju iz datoteke i redom izvode kao da su zadavane tipkanjem." Korisnici i administratori operacijskog sustava koriste skripte za automatiziranje repetetivnih poslova. Značenje termina "skripta" je kasnije prošireno na "niz instrukcija ostalih jezika npr. `awk` skripta, Perl skripta, etc." (Barron, 2000, 4), odnosno na "tekst datoteku, koja je namijenjena izravnom izvršavanju, umjesto prevođenju" (Barron, 2000, 4).

4. Pragmatika skripnih jezika

Naredbe u okruženju ljuske operacijskog sustava Unix čitaju podatke sa standardnog ulaza – tipkovnice, a izlazne rezultate prikazuju na standardnom izlazu – zaslonu (Peek et al, 1998, 67); te komponente poznaju samo jednu vrstu podataka – tekst, odnosno znakovni niz. Tehnikom ulančavanja u ljusci operacijskog sustava Unix – "povezivanje dvije naredbe skupa tako da izlazni podaci iz prve budu ulazni u drugu" (Peek et al, 1998) – moguće je ostvarivati jednostavna funkcionalna programska rješenja – naloge – uz minimum napisanog koda.

Primjer 6 pokazuje ulančavanje naredbi u ljusci operacijskog sustava Unix da bi se ostvario nalog za prikaz broja datoteka koje u nazivu imaju traženi niz znakova:

Primjer 6. Sastavljena naredba koja pokazuje broj datoteka, koje u nazivu imaju niz znakova "txt"

```
$ ls | grep "txt" | wc -l
```

Način povezivanja gotovih komponenti u Primjeru 6 napravljen je primjenom jednostavne sintakse, kojom se na jednostavan način izražava pozivanje komponente i proslijeđivanje podataka u pozvanu komponentu – primjenom znaka "**|**". Upravo je to jedna od osobina bitnih za skriptne jezike, na kojoj se temelje njihove integracijske sposobnosti – *jednostavnost sintakse* za pozivanje gotovih objekata i proslijeđivanje podataka od jednog prema drugom objektu. Nadalje, da bi se podaci mogli proslijeđivati iz jedne komponente (objekta) u drugu, trebaju imati *standardizirani format podataka* za sve komponente. U okruženju ljuske operacijskog sustava Unix, podaci se proslijeđuju u obliku *znakovnih nizova* – na taj način je ostvarena simplifikacija prijenosa podataka i njihova unifikacija.
Skriptni jezik `awk` također ima mogućnosti izdvojiti dio podataka i predati ih, kao znakovni niz, drugim komponentama operacijskog sustava (Aho et al, 1988, 64) – Primjer 7 (unutar `awk` skripte ulančani su naredba `print` i nareba `sort` iz okruženja ljuske):

Primjer 7. Awk skripta, u retku za upis naredbi, koja izdvaja 1. stupac popisa korisnika sustava i poreda podatke po redoslijedu

```
$ who | awk '{ print $1 | "sort" }' > users.txt
```

Dakle, `awk` nema implementiran algoritam za sortiranje redaka teksta ali njegova sintaksa daje podršku za integraciju s vanjskom komponentom sustava s takvom funkcionalnosti.
Znak "**|**" semantički ukazuje na proslijeđivanje, a "**>**" na preusmjeravanje; takva sintaksa je jednostavna za pisanje koda i slikovito ukazuje na značenje.
Na primjerima u skriptnim jezicima `awk` i `bash`, pokazane su integracijske sposobnosti koje posjeduju i ostali skriptni jezici. Mogućnost *integriranja gotovih komponenti* u nekom okruženju presudno je značajna komponenta za rapidni razvoj programskih rješenja.

5. Značaj skripti i skripnih jezika za administraciju operacijskog sustava

Posao administratora operacijskog sustava uključuje ponavljajuće radnje s objektima operacijskog sustava npr. prijepise datoteka iz raznih imenika u jedan određeni imenik, generiranje izvještaja iz logova o događajima u sustavu (engl. event logs) (Stanek, 2003, 3), etc. U svakodnevnom ponavljanju takvih radnji – bilo primjenom miša ili sličnog uređaja u redu s pomagallima u slikovnom sučelju, bilo vlastoručnim upisom i izvršavanjem naredbi, prema određenom redoslijedu – mogu se dogoditi pogreške u izvođenju poslova administracije. Prema Principu automatizacije preporuča se “automatizirati mehaničke i dosadne poslove u kojima se može pogriješiti” (MacLennan, 1999, 10). Rješenje za ovaj problem je izrada i primjena programskih rješenja u skriptnim jezicima; u njih će se pohraniti naredbe i programski konstrukti, koji će izvoditi korake poslova administriranja.

Dakle, aliasi i skripte su, gdjegod su primjenjivi, sredstvo za *automatizaciju* (Stanek, 2003, 3) jednog dijela poslova sustavskog administratora. Automatizacijom repetitivnih poslova, administrator stvara uštedu vremena; oslobođeno vrijeme može posvetiti edukaciji ili kreativnim poslovima, npr. programiranje skripti ili konfiguriranje operacijskog sustava.

Popratna beneficija kreiranja aliasa i skripata je *dokumentiranje* koraka nekog posla u administraciji operacijskog sustava. Nadalje, izvršavanje naloga, aliasa i skripata u retku za upis naredbi ostavlja podatak o izvršenju istih u sustavskim logovima (primjerice *history* u Unix okruženju).

Administratori operacijskog sustava mogu imati znanje o programiranju, ali programiranje nije njihov prvenstveni posao. Njihova prioritetna zaduženja su instalacija i konfiguriranje operacijskih sustava i softvera, otvaranje korisničkih računa, pohrana i povrat podataka, etc. (Collings, 2005, 3). Budući da za te poslove trebaju kontinuirano akumulirati znanja, programski jezici, za primjenu u administraciji sustava, trebali bi imati jednostavnu sintaksu i semantiku, te pragmatički biti primjereni za jednostavnu primjenu i rapidnu izradu programskih rješenja, uz minimum napisanog koda. Skriptni jezici, poput `awk`-a i `bash`-a, zadovoljavaju ovaj zahtjev.

Ne samo da se skriptama može automatizirati administracija sustava (Stanek, 2003, 3), nego se uz pomoć skriptnih jezika mogu generirati nove skripte i na taj način izbjeći mukotrpno tipkanje naredbi u skriptu. Posljednji primjer u članku, Primjer 8, ostvaren u programskom jeziku `awk`, pokazuje jednostavnu skriptu koja će od popisa datoteka napraviti naredbe za njihovo kopiranje u datoteke s nastavkom `.old` u imeniku `/arch`:

Primjer 8. Skripta za generiranje izvornog koda skripte za kopiranje datoteka u zadani imenik

```
#!/bin/bash
ls | awk '{ print "cp " $0 " /arch/" $0 ".old" }' > cp2arch.sh
```

Za svaku naziv datoteke koji prikaže naredba `ls`, u izlaznoj će se datoteci generirati naredba npr. `cp dat1 /arch/dat1.old`.

6. Zaključak

Skriptni programski jezici omogućuju izradu minimalističkih programskih rješenja uz pomoć kojih možemo ostvariti automatizaciju poslova administracije operacijskog sustava. Popratne beneficije u ostvarivanju takvih programskih rješenja su dokumentiranje koraka posla u administraciji te zapisi u sustavskim logovima prilikom njihovog poziva na izvršavanje. Skriptni programski jezici omogućuju rapidni razvoj programskih rješenja za automatizaciju administracije operacijskih sustava, uz minimun uporabljenog programskog koda, jer njihov dizajn podržava razvoj programskih rješenja od gotovih komponenti u razvojnom okruženju, a to su naredbe operacijskog sustava. Dizajn takvih jezika ima simplificiranu sintaksu i semantiku te unificiranu vrstu podataka – znakovni niz. Takve osobine čine ih prihvatljivim za primjenu od strane sustavskih administratora.

Literatura

Aho, Alfred V., Kernighan, Brian W., Weinberger, Peter J. Awk – A Pattern Scanning and Processing Language. Bell Laboratories, 1978.

Aho, Alfred V., Kernighan, Brian W., Weinberger, Peter J. The AWK Programming Language. Addison-Wesley, 1988.

Barron, David. The World of Scripting Languages. Willey, 2000.

Collings, Terry, Wall, Kurt. Red Hat Linux Networking and System Administration. Wiley, 2005.

Kernighan, Brian W., Pike Rob. The UNIX Programming Environment. Prentice Hall, 1984.

Kernighan, Brian W., Ritchie, Dennis M. The C Programming Language, 2nd Edition. Prentice Hall, 1988.

MacLennan, Bruce J. Principles of Prog.ramming Languages: Design, Evaluation, and Implementation. Oxford University Press, 1999.

Newham, Cameron. Learning the bash Shell, 3rd Edition. O'Reilly, 2005.

Ousterhout, John K. Scripting: Higher Level Programming for the 21st Century. Sun Microsystems Laboratories, 1997.

Peek, Jerry, Tondingo, Grace, Strang, John. Learning the UNIX Operating System, 4th Edition. O'Reilly, 1998.

Robbins, Arnold. sed & awk Pocket Reference. O'Reilly, 2000.

Robbins, Arnold. Effective awk Programming, 3rd Edition. O'Reilly, 2001.

Robbins, Arnold, Beebe, Nelson H. F. Classic Shell Scripting. O'Reilly, 2005.

Robbins, Arnold, Lamb, Linda. Learning the vi Editor. O'Reilly, 1998.

Slonneger Kenneth, Kurtz Barry L. Formal Syntax and Semantics of Programming Languages. Addison-Wesley, 1995.

Stanek, William R. Microsoft Windows Server 2003 Administrator’s Pocket Consultant. Microsoft Press, 2003.
Torres, Jesse M. Windows Admin Scripting Little Black Book, 2nd Edition. Paraglyph Press, 2004.