Pogosto zastavljena vprašanja o Unixu

Ted Timar <tmatimar@isgtec.com> v2.9, 11. junij 1996

Pred vami je slovenski prevod spisa Unix FAQ, pogosto zastavljenih vprašanj z odgovori v novičarskih skupinah `comp.unix.questions` in `comp.unix.shell`. Prosim, ne ponavljajte teh vprašanj, ker so bila zastavljena že velikokrat, in prosim, ne pljuvajte po ljudeh, ki morda niso videli tega obvestila. Hvala!

Kazalo

1 Administrativni podatki

1.1 O tem spisu

Pred vami je slovenski prevod spisa Unix FAQ, pogosto zastavljenih vprašanj z odgovori v novičarskih skupinah `comp.unix.questions` in `comp.unix.shell`. Prosim, ne ponavljajte teh vprašanj, ker so bila zastavljena že velikokrat, in prosim, ne pljuvajte po ljudeh, ki morda niso videli tega obvestila. Hvala!

Čeprav so vsa vprašanja legitimna, kaže, da se vsako leto znova pojavljajo v novičarskih skupinah `comp.unix.questions` in `comp.unix.shell`, navadno z odgovori, od katerih je le nekaj pravilnih, in skupaj z jadikovanjem o tem, kako so venomer postavljena enaka vprašanja. Ob tem boste morda želeli prebrati mesečni članek „`Answers to Frequently Asked Questions`“ v novičarski skupini `news.announce.newusers`, ki vam bo povedal, kaj pomeni „UNIX“.

Pri pestrosti sistemov Unix na svetu je težko zagotoviti, da bodo ta vprašanja delovala na vseh sistemih. Vselej preberite strani vašega lokalnega referenčnega sistemskega priročnika, preden poskušate storiti karkoli tukaj priporočenega. Če imate predloge ali popravke za katerikoli tukajšnji odgovor, ga, prosim, pošljite na `tmatimar@isgtec.com`.

Slovenski prevod je nastal konec leta 1998, ko so nekateri odgovori že zastareli. Prevajalec svetuje, da nasvete uporabljate z zrnom soli. Avtor originalnih Unix FAQ načrtuje novo izdajo, ki jo bomo poskušali prevesti, če boste pokazali zanimanje za tole različico. Prevajalec prosi za kakršnekoli pripombe k prevodu - pošljite jih na `roman.maurer@fmf.uni-lj.si`.

1.2 Pravna zaščita

Zbirko originalnih nasvetov je pravno zaščitil © 1994 Ted Timar, razen dela 7 („Pregled različic Unixa“), ki sta ga zaščitila © 1994, Pierre Lewis in Ted Timar. Vse pravice so pridržane. Dovoljenje za razširjanje te zbirke imate, če jo brezplačno razširjate v elektronski obliki, če ste naredili vse razumne poskuse, da bi uporabili njeno zadnjo različico, če so hranjene vse tu navedene zasluge in to sporočilo o pravni zaščiti. O drugih vprašanjih distribucije se bo premislilo. Vsem razumnim zahtevam se bo ugodilo.

Vsi nasveti so bili priporočeni z dobrimi nameni, vendar za njihovo točnost ne jamčijo niti pisci prispevkov niti avtor te zbirke niti prevajalec. Uporabniki v celoti prevzemajo odgovornost za morebitno povzročeno škodo.

Slovenski prevod te zbirke z dne 26. septembra 1998 je delo Romana Maurerja. Za njegovo razširjanje veljajo enaki pogoji kot za izvirno zbirko.

1.3 Dostopnost

Veliko zbirk pogosto zastavljenih vprašanj (krajše PZV; angl. FAQ - Frequently Asked Questions), vključno s to, je dostopnih z anonimnim FTP-jem z arhivarskega mesta `<ftp://rtfm.mit.edu/pub/usenet/news.answers/>`, ki je zrcaljen tudi na slovenskem strežniku `<ftp://ftp.arnes.si/usenet/news.answers/>`. Ime, pod katerim se arhivira PZV, vidite v vrstici „`Archive-Name:`“ na vrhu članka. Original teh PZV je shranjen pod imeni „`unix-faq/faq/part[1-7]`“.

Formatirane različice originalnega spisa Unix FAQ v formatu Texinfo, tekstovnem formatu in formatu za `*roff` so dostopne z anonimnim FTP-jem v imeniku `<ftp://ftp.wg.omron.co.jp/pub/unix-faq/docs/>`.

Slovenska različica v formatih TXT, SGML, Postscript in DVI je dostopna na mestu za anonimni FTP v podimenikih `<ftp://ftp.lugos.si/pub/lugos/doc/FAQ-sl/>` (iščite datoteke `Unix-FAQ-sl.*`) ali v svetovnem spletu na naslovu `<http://www.lugos.si/delo/slo/HOWTO-sl/Unix-FAQ-sl.html>`.

1.4 Zahvale

```
Subject: Who helped you put this list together?
Date: Thu Mar 18 17:16:55 EST 1993
```

Ta spis je ena prvih zbirk pogosto zastavljenih vprašanj (PZV; angl. Frequently Asked Questions - FAQ). Prvič je bil sestavljen julija 1989.

Ted Timar <tmatimar@isgtec.com> je prevzel vzdrževanje tega seznama. Skoraj vse delo (in zasluge) za sestavljanje te zbirke gredo Stevu Haymanu.

Veliko zahval dolgujemo tudi ducatom bralcev Useneta. Ti so prispevali vprašanja, odgovore, popravke in predloge za ta seznam. Posebna zahvala gre Maartenu Litmaathu, Guyu Harrisu in Jonathanu Kamensu, ki so spisali posebej obsežne prispevke.

Temo 6 („Vprašanja, povezana z različnimi ukaznimi lupinami in njihovimi razlikami") je pretežno zapisal Matthew Wicks <wicks@dcdmjw.fnal.gov>.

Temo 7 („Pregled različic Unixa") je pretežno zapisal Pierre (P.) Lewis <lew@bnr.ca>.

Kjer je le mogoče, je na vrhu odgovora naveden avtor posameznega odgovora na vprašanje in datum zadnjega popravka. Žal sem s to prakso začel šele nedavno, zato je veliko podatkov nepopolnih. Bil sem tudi brezbrižen do podatkov o avtorjih popravkov odgovorov. Opravičujem se vsem, ki ste napisali dragocene prispevke, a niste primerno omenjeni.

Originalni dokument je dostopen v formatu `*roff` (paketi makro ukazov `ms` in `mm`). Andrew Cromarty ga je pretvoril v format Texinfo. Marty Leisner <leisner@sdsp.mc.xerox.com> je očistil različico v formatu Texinfo.

Pisci obsežnejših prispevkov tega spisa so lahko omenjeni drugje ali pa tudi ne, a so:

- Steve Hayman <shayman@Objectario.com>
- Pierre Lewis <lew@bnr.ca>
- Jonathan Kamens <jik@mit.edu>
- Tom Christiansen <tchrist@mox.perl.com>
- Maarten Litmaath <maart@nat.vu.nl>
- Guy Harris <guy@auspex.com>

Dne 26. septembra 1998 s popravki 16. februarja 1999 je Roman Maurer <roman.maurer@fmf.uni-lj.si> ta spis prevedel v slovenščino. Hvala Alešu Koširju <kosir@hermes.si> za številne popravke prevoda!

2 Splošna vprašanja

2.1 Kaj pomeni številka v oklepajih pri sklicevanju na ‚`rm(1)`' ali ‚`ctime(3)`'?

```
Subject: When someone refers to 'rn(1)' ...  the number in parentheses mean?
Date: Tue, 13 Dec 1994 16:37:26 -0500
```

Številka je videti kot neke vrste klic funkcije, a to ni. Te številke pomenijo poglavje „Priročnika za Unix" (angl. „Unix manual"), kjer najdete primerno dokumentacijo. Napišete lahko

```
man 3 ctime
```

in s tem pogledate stran o „`ctime`“ v tretjem poglavju priročnika.

Tradicionalna poglavja priročnika so:

1

Ukazi na uporabniškem nivoju

2

Sistemski klici

3

Funkcije v knjižnicah

4

Naprave in gonilniki naprav

5

Datotečni formati

6

Igre

7

Različne zadeve - paketi makro ukazov itd.

8

Ukazi za vzdrževanje in tek sistema

Nekatere različice Unixa uporabljajo neštevilska imena razdelkov. Xenix, na primer, uporablja „`C`“ za ukaze in „`S`“ za funkcije. Nekatere novejše različice Unixa se razlikujejo in razumejo ukaz v obliki „`man -s# naslov`“ namesto „`man # title`“.

Vsako poglavje ima uvod, ki ga lahko preberete z ukazom

```
man # intro
```

kjer je `#` številka poglavja.

Številka je včasih potrebna, da lahko ločite med ukazom in funkcijo v knjižnici ali sistemskim klicem z enakim imenom. Vaš sistem ima na primer morda „`time(1)`“, stran referenčnega priročnika o ukazu ‚`time`‘ za merjenje časov izvajanja programov, in tudi „`time(3)`“, stran referenčnega priročnika o podprogramu ‚`time`‘, s katerim prikažete trenutni čas. Uporabite lahko različici „`man 1 time`“ ali „`man 3 time`“ in s tem določite, katera stran priročnika vas zanima.

Pogosto boste našli druge razdelke za lokalne programe ali celo podrazdelke zgornjih razdelkov - Ultrix ima med drugim razdelke 3m, 3n, 3x in 3yp.

2.2 Kaj pomeni {kakšen nerazumljiv ukaz na Unixu}?

```
Subject: What does {some strange unix command name} stand for?
Date: Thu Mar 18 17:16:55 EST 1993
```

awk

"Aho Weinberger & Kernighan"

Jezik in program, ki ga interpretira, se imenujeta po začetnicah priimkov njegovih avtorjev, ki so Al Aho, Peter Weinberger in Brian Kernighan.

grep

"Global Regular Expression Print"

Izraz `grep` izhaja iz ukaza za `ed`, ki je izpisal vse vrstice, ki so ustrezale podanemu iskalnemu vzorcu

```
g/re/p
```

kjer pomeni „`re`“ „regularni izraz“ (angl. regular expression).

fgrep

"Fixed GREP"

`fgrep` išče le fiksne vzorce. Črka ‚`f`‘ ni okrajšava za „fast“ (slov. „hitro“) - pravzaprav je „`fgrep foobar *.c`“ navadno počasnejši kot „`egrep foobar *.c`“ (Da, to je nekam presenetljivo. Poskusite!)

`Fgrep` je vseeno uporaben, ko iščete v datoteki večje število iskalnih vzorcev, kot jih prenese `egrep`.

egrep

"Extended GREP"

`egrep` uporablja imenitnejše regularne izraze kot `grep`. Mnogi ljudje vselej uporabljajo `egrep`, saj uporablja boljše algoritme kot `grep` ali `fgrep` in je navadno najhitrejši od njih.

cat

ČATenate"

Obrobna angleška beseda „catenate“ pomeni „spenjati (z verigo)“ ali „povezovati v zaporedja“, kar je točno to, kar naredi „`cat`“ z eno datoteko ali z več datotekami. Ne zamenjujte ga s C/A/T, (angl. Computer Aided Typesetter, slov. računalniško podprti pisalni stroj).

gecos

"General Electric Comprehensive Operating Supervisor"

Ko je podjetje General Electric prodalo svojo enoto za velike sisteme podjetju Honeywell, je ta izpustil „`E`“ iz „`GECOS`“.

Datoteka z gesli za Unix ima polje „`pw_gecos`“. To polje je resnični ostanek zgodnjih časov. Dennis Ritchie je poročal:

> Včasih smo poslali tiskalniški izhod ali paketna opravila na stroj GCOS. Polje `gcos` v datoteki z gesli je bilo mesto, v katerega smo stlačili podatke za kartico `$IDENT`. Ni ravno elegantno.

nroff

"New ROFF"

troff

"Typesetter new ROFF"

To sta naslednika programa „`roff`“, ki je bil ponovna izvedba programa „`runoff`“ s sistema Multics in s katerim ste „iztirili“ (angl. „run off“) dober izvod vašega dokumenta.

tee

T

Po izrazoslovju vodovodnih inštalaterjev - cepilec pipe v obliki črke T.

bss

"Block Started by Symbol"

Dennis Ritchie pravi:

> Pravzaprav ta akronim (tako kot smo ga vzeli; morda ima druge verjetne etimologije) pomeni „Blok se je začel s simbolom“ (angl. „Block Started by Symbol“). To je bila psevdo-operacija v FAP (Fortran Assembly [-hm?] Program), zbirnik za stroje IBM 704-709-7090-7094. Definiral je svoje oznake in rezerviral prostor za dano število besed. Obstajal je še psevdo-operator BES, „Block Ended by Symbol“ (slov. „Blok se je končal s simbolom“), ki je počel podobno, a tako, da je bila oznaka definirana z zadnjo prirejeno besedo + 1. (Na teh strojih so bile fortranske tabele shranjene v obratnem vrstnem redu in so bile 1-začetek.) Uporaba je razumno primerna in je podobna kot pri standardnem nalagalniku Unixa, kjer prirejeni pomnilnik ni potrebno vstaviti dobesedno v objekt, pač pa je dostopen prek kazalca (angl. „the space assigned didn't have to be punched literally into the object deck but was represented by a count somewhere“).

biff

"BIFF"

Ta ukaz, ki vklopi asinhrono obvestilo o pošti, je bil pravzaprav poimenovan po psu v Berkeleyu. Eric Cooper z univerze Carnegie Mellon takole pripoveduje:

> Potrdim lahko izvor imena `biff`, če vas zanima. Biff je bilo ime psu Heidi Stettner, ko je bila Heidi (in jaz in Bill Joy) študentka na U. C. Berkeley in so razvijali zgodnje verzije BSD. Biff je bil popularen med stanovalci Evans Hall in je bil znan po tem, da je vselej lajal na poštarja. Od tod ime tega ukaza.

rc (kot v „`.cshrc`“ ali „`/etc/rc`“)

"RunCom"

„`rc`“ izhaja iz „`runcom`“ s sistema MIT CTSS približno okrog leta 1965. Brian Kernighan & Dennis Ritchie sta takole pripovedovala Vicki Brown:

> Obstajal je pripomoček, ki je izvedel ukaze, shranjene v datoteki; imenoval se je „`runcom`“, okrajšava za „run commands“ (slov. poženi ukaze), in takšno datoteko smo začeli klicati „neki runcom“. „`rc`“ v Unixu je ostalina te uporabe.

„`rc`“ je tudi ime ukazne lupine novega operacijskega sistema Plan 9.

Perl

"Practical Extraction and Report Language" ali

Perl

"Pathologically Eclectic Rubbish Lister"

Jezik Perl je zelo popularno orodje avtorja Larryja Walla. Je prosto dostopno in popolnoma prenosljivo. Uporablja se za obdelovanje besedil in delo z datotekami, ter premošča prepad med ukazno lupino in programiranjem v jeziku C (ali med preprostim ukazom iz ukazne vrstice in puljenjem svojih las). Za nadaljnje informacije glejte novičarsko skupino Useneta `comp.lang.perl.misc`.

Knjiga „Life with Unix“ Dona Libesa vsebuje mnogo več takšnih poslastic.

2.3 Kako deluje most med novičarsko skupino „comp.unix.questions“ in elektronskim seznamom „info-unix“?

```
Subject: How does the gateway between "comp.unix.questions" ... work ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Elektronska seznama „info-unix“ in „unix-wizards“ sta posebni poštni različici skupin `comp.unix.questions` in `comp.unix.wizards`. Med vsebino elektronskih poštnih seznamov in novičarskih skupin ne bi smelo biti razlik.

Če se želite prijaviti na enega teh seznamov, pošljite pismo na `info-unix-request@brl.mil` ali `unix-wizards-request@brl.mil`. Ne pričakujte takojšnjega odgovora.

Tukaj so podrobnosti, zahvaljujoč vzdrževalcu seznama, Bobu Reschlyju:

2.3.1 Pošiljanje na seznama info-UNIX in UNIX-wizards

Vse, kar pošljete na seznam, se objavi; pisem ne moderiram - BRL deluje le kot prepisovalnik. Prispevki naročnikov z Interneta naj bodo poslani na naslov seznama (info-UNIX ali UNIX- wizards); naslova z ‚-request‘ sta le za dopisovanje z vzdrževalcem seznama [mano]. Pisma, poslana prek Useneta, naj bodo naslovljene na primerno novičarsko skupino (`comp.unix.questions` ali `comp.unix.wizards`).

Naročniki z Interneta dobivajo pisma dveh vrst: posamezna sporočila in povzetke pisem. Pisma, ki prispejo na BRL prek Interneta in BITNET-a (čez most BITNET-Internet), se takoj razpošljejo na vse naslove na poštnem seznamu. Pisma z Useneta se zberejo v povzetke in ti se dnevno pošljejo na naslove članov na seznamu.

Promet z BITNET-a je precej podoben prometu prek Interneta. Glavna razlika je, da vzdržujem le en naslov za pisma, naslovljena na vse naročnike BITNET-a. Ta naslov kaže na večji seznam, ki vsebuje posamezne naročnike BITNET-a. Tako mora v vsaki smeri le po eno pismo med BITNET-om in Internetom.

Naročniki USENET-a vidijo le posamezna pisma. Vsa pisma, odposlana z Internetu, se razpošiljajo na naš stroj z USENET-om. Potem se uvrstijo v primerno novičarsko skupino. Žal pri sporočilih, ki gredo prek mostu, pošiljatelj postane „`news@brl-adm`“. To je trenutno neizbežni stranski učinek premostitvenega programja.

Kar se tiče publike, USENET ima izjemno veliko bralcev - domnevam, da več tisoč gostiteljev in na desetine tisočev bralcev. Glavni seznam, ki ga vzdržujemo tu na BRL, poganja okoli 150 vnosov, od katerih jih je približno 10% lokalnih redistribucijskih seznamov. Nimam natančnega občutka za velikost redistribucije po BITNET-u, a domnevam, da je približno enake velikosti in sestave kot glavni seznam. Promet vsebuje v povprečju od 150 Kb do 400 Kb sporočil na teden.

2.4 Katere so uporabne knjige o Unixu ali C-ju?

```
Subject: What are some useful Unix or C books?
Date: Thu Mar 18 17:16:55 EST 1993
```

Mitch Wright (`mitch@cirrus.com`) vzdržuje dober seznam knjig o Unixu in C-ju, z opisi in z nekaterimi kratkimi pregledi. Trenutno je na njegovem seznamu 167 knjižnih naslovov.

Kopijo tega seznama dobite z anonimnim FTP-ju na naslovu `<ftp://ftp.rahul.net/pub/mitch/YABL/yabl>` (192.160.13.1). Popravke in predloge pošiljajte na `mitch@cirrus.com`.

Samuel Ko (`kko@sfu.ca`) vzdržuje še en seznam knjig za Unix. Ta seznam vsebuje le priporočene knjige in je torej krajši, knjige pa so razvrščene po kategorijah, kar je morda primernejše, če iščete določen tip knjige.

Kopijo tega seznama dobite na naslovu `<ftp://rtfm.mit.edu/pub/usenet/news.answers/books/unix>`. Popravke in predloge pošiljajte na `kko@sfu.ca`.

Če ne morete uporabiti anonimnega FTP-ja, pošljite vrstico „`help`“ na `ftpmail@decwrl.dec.com` za navodila o prenašanju datotek po elektronski pošti.

2.5 Kaj se je zgodilo s seznamom izgovarjave, ki je bil del tega spisa?

```
Subject: What happened to the pronunciation list ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Od nastanka v letu 1989 dalje so originalna angleška PZV vključevala izčrpen seznam izgovarjave, ki ga je vzdrževal Maarten Litmaath (hvala, Maarten!). Prvi seznam je pripravil Carl Paukstis <`carlp@frigg.isc-br.com`>.

Seznam se je upokojil, saj njegov naslov „Vprašanja o Unixu“ ne ustreza. Še vedno ga lahko najdete kot del široko razširjene datoteke „Jargon“ (vzdržuje jo Eric S. Raymond, <`eric@snark.thyrsus.com`>), ki je veliko primernejši forum na temo „Kako naj izgovorim /* ¿‘.

Če želite kopijo seznama, se povežite z 133.210.4.4, in jo poiščite v imeniku <`ftp://ftp.wg.omron.co.jp/pub/unix-faq/docs/Pronunciation-Guide`>.

3 Relativno osnovna vprašanja, ki jih navadno sprašujejo začetniki

3.1 Kako odstranim datoteko z imenom, ki se začne z ‚–‘?

```
Subject: How do I remove a file whose name begins with a "-" ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Poiščite tak način imenovanja datoteke, da se ne začne s pomišljajem. Najpreprostejši odgovor je primer

```
rm ./-imedatoteke
```

(seveda predvidevamo, da je „`-imedatoteke`“ v tem primeru v trenutnem imeniku).

Ta način izogibanja dvoumnemu ‚`-`‘ na začetku imen deluje tudi pri drugih ukazih.

Veliko ukazov, posebej tistih, ki so bili napisani z uporabo rutine „`getopt(3)`“ za razčlembo argumentov, sprejme tudi argument „–“, ki pomeni „to je zadnja izbira, vse po tem ni več izbira“. Vaša različica `rm` potemtakem morebiti razume tudi

```
rm -- -imedatoteke
```

Nekatere različice `rm`, ki ne uporabljajo `getopt()`, razumejo podobno tudi en sam „–“, torej lahko poskusite kar z „`rm - -imedatoteke`“.

3.2 Kako odstranim datoteko, ki ima v svojem imenu čudne znake?

```
Subject: How do I remove a file with funny characters in the filename ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Če je „čuden znak“ znak ‚`/`‘, preskočite na zadnji del odgovora. Če je čuden znak nekaj drugega, na primer presledek ali kontrolni znak ali znak s prižganim osmim bitom (to so npr. slovenske črke „`čšžČŠŽ`“), nadaljujte z branjem.

Prvi klasični odgovor je uporaba ukaza na način

```
rm -i vzorec*ki*ustreza*le*datoteki*ki*jo*želite
```

Ukaz vpraša za vsako datoteko, ki ustreza vzorcu, če jo želite odstraniti. Nasvet morda v vaši ukazni lupini ne bo deloval zaradi znakov s prižganim osmim bitom, saj ga ukazna lupina lahko odreže.

Drugi klasični odgovor je

```
rm -ri .
```

ki vas za vsako datoteko v trenutnem imeniku vpraša, ali naj jo odstrani.

V angleškem okolju odgovorite z ‚y' pri problematični datoteki in z ‚n' pri vseh ostalih datotekah. Žal to ne deluje pri veliko različicah rm. Hkrati se bo ukaz izvedel za vse datoteke na vseh podimenikih trenutnega imenika, torej boste morda začasno hoteli te imenike narediti nedostopne z „chmod a-x", da vas ne bodo motili.

Vselej globoko vdihnite in razmislite, kaj počnete, ter dvakrat preverite, kaj ste zapisali v ukazni vrstici, kadar uporabljate zastavico „-r" ukaza rm ali znaka * ali ? (angl. wildcard).

Tretji klasični odgovor je

```
find . -type f ... -ok rm '{}' \;
```

kjer je „..." skupina oznak, ki enolično določa ime datoteke. Ena od možnosti je, da za problematično datoteko poiščete njeno številko inode (z uporabo ukaza „ls -i .") in potem uporabite

```
find . -inum 12345 -ok rm '{}' \;
```

ali

```
find . -inum 12345 -ok mv '{}' novo-ime-datoteke \;
```

„-ok" je varnostno preverjanje - program vas bo vprašal za potrditev ukaza, preden ga bo izvedel. Spraševanju se lahko izognete z uporabo „-exec", če radi živite nevarno, ali če domnevate, da ime datoteke vsebuje zaporedje čudnih znakov, ki vam bodo ob izpisu zmešali zaslon.

Kaj, če ime datoteke vsebuje znak ‚/'?

Te datoteke so res posebni primeri in jih lahko ustvari le hroščata koda v jedru (tipično jih naredijo implementacije NFS, ki ne počistijo neveljavnih znakov iz imen datotek na oddaljenih strojih). Najprej poskušajte natanko razumeti, zakaj je ta problem tako izjemen.

Spomnite se, da so imeniki v Unixu preprosto pari imen datotek in številk inode. Imenik tipično vsebuje podobno informacijo:

```
ime datoteke  inode

datoteka1     12345
datoteka2.c   12349
datoteka3     12347
```

Teoretično sta ‚/' in ‚\0' le dva znaka, ki se ne moreta pojaviti v imenu datoteke - znak ‚/' zato, ker se uporablja za ločevanje imenikov in datotek, in znak ‚\0' zato, ker končuje ime datoteke.

Žal nekatere izvedbe omrežnega datotečnega sistema NFS prešerno ustvarjajo datoteke z vključenimi nagibnicami kot odgovor na zahteve oddaljenih strojev. Na primer, to se lahko zgodi, ko se nekdo na Macu ali drugem ne-Unixu odloči narediti oddaljeno datoteko NFS na vašem Unixu, z datumom v imenu datoteke. Vaš imenik na Unixu ima potem v sebi tole:

```
ime datoteke  inode

91/02/07      12357
```

Takšne datoteke ne bo pobrisalo nobeno zgoraj opisano čaranje z ukazoma `find` in `rm`, saj morata tako tadva pripomočka kot tudi vsi drugi programi na Unixu razumeti znak ‚/' na običajen način.

Vsak običajen program bo želel narediti `unlink("91/02/07")`, kar jedro razume kot „pobriši datoteko `07` v podimeniku `02` imenika `91`", toda to ni tisto, kar hočemo - imamo **datoteko** imenovano „`91/02/07`" v trenutnem imeniku. To je komaj zaznavna, a ključna razlika.

Kaj lahko naredite v tem primeru? Najprej se poskusite vrniti na Maca, ki je ustvaril ta nevšečni vnos, in skušajte prepričati Maca in vaš lokalni strežnik NFS, da datoteko preimenujeta v nekaj, kar ne vsebuje nagibnic.

Če to ne deluje ali morda ni izvedljivo, boste potrebovali pomoč vašega sistemskega upravitelja, ki bo moral poskusiti kaj od naslednjega. Uporabite „`ls -i`", da boste izvedeli številko inode nesrečne datoteke, potem odklopite datotečni sistem (z `umount`), uporabite „`clri`" za izbris inodea, in popravite datotečni sistem s „`fsck`" in s stisnjenimi pestmi. Postopek uniči podatke o datoteki. Če jih želite obdržati, lahko poskusite:

- ustvarite nov imenik v istem starševskem imeniku, kot ga ima imenik, ki vsebuje datoteko na slabem glasu,
- iz starega imenika v novega premaknite vse, kar lahko (se pravi vse, razen datoteke s slabim imenom),
- na imeniku z datoteko s slabim imenom napravite „`ls -id`" , da dobite njegovo številko inumber,
- odklopite datotečni sistem,
- uporabite „`clri`" na imeniku, ki vsebuje sporno datoteko,
- obnovite datotečni sistem s „`fsck`".

Potem boste datoteko poiskali takole:

- namestite datotečni sistem;
- preimenujte imenik, ki ste ga ustvarili, tako da bo imel enako ime, kot stari imenik (saj je stari imenik dokončno pobrisal „`fsck`");
- premaknite datoteko iz imenika „`lost+found`" v imenik s primernejšim imenom.

Alternativa temu pristopu je popravljanje imenika na zahteven način z brkljanjem po surovem datotečnem sistemu (angl. raw file system). Uporabite „`fsdb`", če ga imate.

3.3 Kako dobim rekurziven izpis vsebine imenika?

```
Subject: How do I get a recursive directory listing?
Date: Thu Mar 18 17:16:55 EST 1993
```

Nekaj od tega bo naredilo to, kar želite:

```
ls -R               (vse različice ‚ls' nimajo ‚-R')
find . -print       (to mora delovati povsod)
du -a .             (pokaže ime in velikost)
```

Če iščete vzorec z jokerji (angl. wildcard), ki bo ustrezal vsem datotekam „`.c`" v tem imeniku in pod njim, ga ne boste našli, pač pa lahko uporabite:

```
% nek-ukaz `find . -name '*.c' -print`
```

„`find`“ je zmogljiv program. Poučite se o njem.

3.4 Kako lahko v svoj pozivnik vključim ime trenutnega imenika?

```
Subject: How do I get the current directory into my prompt?
Date: Thu Mar 18 17:16:55 EST 1993
```

Odvisno od tega, katero ukazno lupino uporabljate. V nekaterih ukaznih lupinah je to preprosto, v drugih pa težko ali celo nemogoče.

- C Shell (`csh`): Vstavite tole v svojo datoteko `.cshrc` - preuredite spremenljivko pozivnika po svojem okusu.

  ```
  alias setprompt 'set prompt="${cwd}% "'
  setprompt          # nastavitev uvodnega pozivnika
  alias cd 'chdir \!* && setprompt'
  ```

  Če uporabljate `pushd` in `popd`, boste potrebovali tudi:

  ```
  alias pushd 'pushd \!* && setprompt'
  alias popd  'popd  \!* && setprompt'
  ```

  Nekatere lupine `csh` nimajo spremenljivke `$cwd` - namesto tega lahko uporabite `` `pwd` ``.

  Če želite dobiti v vaš pozivnik le zadnjo komponento vašega trenutnega imenik („`mail%` “ namesto „`/usr/spool/mail%` “), lahko uporabite

  ```
  alias setprompt 'set prompt="$cwd:t% "'
  ```

  Pomen operatorjev `&&` in || v nekaterih starejših lupinah `csh` je zamenjan. Poskusite:

  ```
  false && echo napaka
  ```

  Če izpiše „`napaka`“, morate zamenjati `&&` in || (in dobiti boljšo različico `csh`.)

- Bourne Shell (`sh`): Če imate novejšo različico Bournove ukazne lupine (SVR2 ali novejšo), lahko uporabite za izdelavo svojega ukaza funkcijo lupine, denimo „`xcd`“:

  ```
  xcd() { cd $* ; PS1="`pwd` $ "; }
  ```

  Če imate starejšo Bournovo ukazno lupino, je to zapleteno, a mogoče. Dodajte v vašo datoteko `.profile`:

  ```
  LOGIN_SHELL=$$ export LOGIN_SHELL
  CMDFILE=/tmp/cd.$$ export CMDFILE
  # 16 je SIGURG, izberite signal, ki se navadno ne uporablja
  PROMPTSIG=16 export PROMPTSIG
  trap '. $CMDFILE' $PROMPTSIG
  ```

  in potem postavite ta izvedljivi skript (brez zamikanja!), recimo mu „`xcd`“ nekam v vašo pot (`PATH`).

  ```
  : xcd directory - change directory and set prompt
  : by signalling the login shell to read a command file
  cat >${CMDFILE?"not set"} <<EOF
  cd $1
  PS1="\`pwd\`$ "
  EOF
  kill -${PROMPTSIG?"not set"} ${LOGIN_SHELL?"not set"}
  ```

Zdaj spreminjajte imeniku z ukazom „`xcd /nek/imenik`“.

- Korn Shell (`ksh`): Vstavite to v vašo datoteko `.profile`:

  ```
  PS1='$PWD $ '
  ```

  Če želite le zadnjo komponento imenika, uporabite

  ```
  PS1='${PWD##*/} $ '
  ```

- T C shell (`tcsh`) `Tcsh` je popularna izboljšana različica `csh` z nekaterimi dodatnimi vgrajenimi spremenljivkami (in veliko drugih zmožnosti):

  ```
  %~          trenutni imenik, uporablja ~ za $HOME
  %/          polna pot trenutnega imenika
  %c ali %.   končna komponenta trenutnega imenika
  ```

  tako, da lahko preprosto ukažete

  ```
  set prompt='%~ '
  ```

- BASH (FSF-jeva ukazna lupina „Bourne Again SHell“) Simbol `\w` v `$PS1` daje polno ime poti trenutnega imenika, z uporabo ˜ za `$HOME`; `\W` daje osnovno ime (angl. basename) trenutnega imenika. Torej lahko poleg zgornjih rešitev za `sh` in `ksh` uporabite tudi

  ```
  PS1='\w $ '
  ```

  ali

  ```
  PS1='\W $ '
  ```

3.5 Kako s skriptom ukazne lupine preberem znake s terminala?

```
Subject: How do I read characters from the terminal in a shell script?
Date: Thu Mar 18 17:16:55 EST 1993
```

V `sh`, uporabite `read`. Navadno se uporabi podobna zanka:

```
while read vrstica
do
    echo $vrstica
    ...
done
```

V `csh` uporabite `$<`, kot tukaj:

```
while ( 1 )
    set vrstica = "$<"
    if ( "$vrstica" == "" ) break
    ...
end
```

Žal se v `csh` ne da razločevati med prazno vrstico in koncem datoteke.

Če uporabljate `sh` in bi radi prebrali **en sam** znak s terminala, lahko poskusite nekaj podobnega:

```
echo -n "Vstavite znak: "
stty cbreak         # ali  stty raw
beriznak=`dd if=/dev/tty bs=1 count=1 2>/dev/null`
stty -cbreak

echo
echo "Hvala, da ste vtipkali $beriznak ."
```

3.6 Kako preimenujem `*.foo` v `*.bar` ali spremenim imena datotek v male črke?

```
Subject: How do I rename "*.foo" to "*.bar", or change file names to lowercase?
Date: Thu Mar 18 17:16:55 EST 1993
```

Zakaj ne deluje „`mv *.foo *.bar`“? Premislite, kako ukazna lupina razvija jokerje (angl. wildcards). Niza „`*.foo`“ in ‚`*.bar`“ se razvijeta, preden ukaz `mv` sploh vidi argumente. Odvisno od vaše ukazne lupine lahko to spodleti na več načinov. `CSH` izpiše „`No match.`“, ker ne najde „`*.bar`“. `SH` izvede „`mv a.foo b.foo c.foo *.bar`“, kar bo uspelo le, če imate en sam imenik, ki ustreza „`*.bar`“, kar je zelo neverjetno in skoraj zagotovo ni tisto, kar ste imeli v mislih.

Odvisno od vaše ukazne lupine lahko to storite z zanko, v kateri poženete „`mv`“ na vsaki posamezni datoteki. Če ima vaš sistem „`basename`“, lahko uporabite:

- C Shell:

```
foreach f ( *.foo )
    set base=`basename $f .foo`
    mv $f $base.bar
end
```

- Bourne Shell:

```
for f in *.foo; do
    base=`basename $f .foo`
    mv $f $base.bar
done
```

Nekatere ukazne lupine poznajo posebne mehanizme za zamenjavo vrednosti spremenljivk (angl. variable substitution features), tako lahko namesto ukaza „`basename`“ uporabite vgrajene mehanizme, kot so:

- C Shell:

```
foreach f ( *.foo )
    mv $f $f:r.bar
end
```

- Korn Shell:

```
for f in *.foo; do
    mv $f ${f%foo}bar
done
```

Če nimate ukaza „`basename`“ ali bi želeli preimenovati `foo.*` v `bar.*`, lahko uporabite kar „`sed`“ ali podobno orodje, da predelate ime datoteke, a v splošnem je ideja zanke enaka. S pripomočkom „`sed`“ lahko spremenite imena datotek v ukaze za `mv` in pošljete ukaze za izvajanje lupini „`sh`“. Poskusite:

```
ls -d *.foo | sed -e 's/.*/mv & &/' -e 's/foo$/bar/' | sh
```

Obstaja program Vladimirja Lanina, imenovan „mmv“, ki opravi preimenovanje datotek in je bil aprila 1990 objavljen v comp.sources.unix (Volume 21, issues 87 and 88). Omogoča vam uporabo na način

```
mmv '*.foo' '=1.bar'
```

V ukaznih lupinah zanke, podobne zgornjim, pomagajo pri zamenjavi imen datotek iz velikih črk v male ali obratno. Velike črke v imenih datotek prevedete v male takole:

- C Shell:

```
foreach f ( * )
    mv $f `echo $f | tr '[A-Z]' '[a-z]'`
end
```

- Bourne Shell:

```
for f in *; do
    mv $f `echo $f | tr '[A-Z]' '[a-z]'`
done
```

- Korn Shell:

```
typeset -l l
for f in *; do
    l="$f"
    mv $f $l
done
```

Če želite biti posebno temeljiti in pravilno obravnavati datoteke z izrojenimi imeni (ki vključujejo presledke ali tako-rekoč karkoli), boste morali uporabiti:

- Bourne Shell:

```
for f in *; do
    g=`expr "xxx$f" : 'xxx\(.*\)' | tr '[A-Z]' '[a-z]'`
    mv "$f" "$g"
done
```

  Ukaz ‚expr‘ vedno izpiše ime datoteke, tudi, če je to enako ‚-n‘, ali, če vsebuje ubežna zaporedja za System V, kakršno je ‚\c‘.

  Nekatere različice ukaza „tr“ potrebujejo ‚[‘ in ‚]‘, nekatere ju ne. V tem konkretnem primeru ju je povsem neškodljivo vključiti. Različice ukaza tr, ki nočejo „[]“ bodo primerno mislile, da bi morale preslikati ‚[‘ v ‚[‘ in ‚]‘ v ‚]‘.

- Če imate nameščen programski jezik „perl“, lahko ta skript Larryja Walla za preimenovanje datotek uporabite zelo koristno. Uporabljate ga lahko za širok spekter sprememb imen datotek.

```
#!/usr/bin/perl
#
# primeri uporabe skripta rename (avtor lwall):
#       rename 's/\.orig$//' *.orig
#       rename 'y/A-Z/a-z/ unless /^Make/' *
#       rename '$_ .= ".bad"' *.f
#       rename 'print "$_: "; s/foo/bar/ if <stdin> =~ /^y/i' *

$op = shift;
for (@ARGV) {
```

```
            $was = $_;
            eval $op;
            die $@ if $@;
            rename($was,$_) unless $was eq $_;
        }
```

3.7 Zakaj dobim ob ukazu „**`rsh gostitelj ukaz`**“ [kakšno čudno sporočilo o napaki]?

```
Subject: Why do I get [some strange error message] when I "rsh host command" ?
Date: Thu Mar 18 17:16:55 EST 1993
```

(Govorimo o programu za oddaljeno ukazno lupino „`rsh`“, ki se ponekod imenuje „`remsh`“ ali „`remote`“. Obstaja tudi ukazna lupina z omejitvami, imenovana „`rsh`“, kar pa je druga stvar.)

Če vaš oddaljeni račun uporablja C-jevsko ukazno lupino (`csh`), bo oddaljeni gostitelj pognal `csh`, da bi izvedel vaš ‚`ukaz`‘, in ukazna lupina bo prebrala vašo oddaljeno datoteko `.cshrc`. Morda ta `.cshrc` vsebuje „`stty`“, „`biff`“ ali kakšen drug ukaz, ki ni primeren za neinteraktivno ukazno lupino. Nepričakovan izhod ali sporočilo o napaki pri teh ukazih lahko zapleteta vašo lupino `rsh` na čudne načine.

Tukaj je primer. Denimo, da imate v vaši datoteki `.cshrc` tole:

```
stty erase ^H
biff y
```

Dobili boste čudna sporočila, kot je tole:

```
% rsh nek-stroj date
stty: : Can't assign requested address
Where are you?
Tue Oct  1 09:24:45 EST 1991
```

Podobne napake lahko dobite, če izvajate določena opravila pripomočkov „`at`“ ali „`cron`“, ki prav tako bereta vašo datoteko `.cshrc`.

Na srečo je popravek preprost. Zelo verjetno je v vaši datoteki „`.cshrc`“ **na kupe** operacij, ki jih preprosto ni vredno izvajati drugje kot v interaktivnih ukaznih lupinah (npr. „`set history=N`“). Tedaj jih postavite v pogojni blok v vaši „`.cshrc`“ z

```
if ( $?prompt ) then
        operacije...
endif
```

Ker v neinteraktivnih ukaznih lupinah spremenljivka „`prompt`“ ni nastavljena, bodo sporne operacije izvedene le v interaktivnih ukaznih lupinah.

Morda boste želeli premakniti nekatere ukaze v vaš prijavni skript `.login`. Če se morajo ti ukazi izvesti le ob vaši prijavi na sistem (iskanje nove pošte, neprebranih novičarskih sporočil in tako naprej), jih je bolje imeti v datoteki `.login`.

3.8 Kako naj {nastavim spremenljivko okolja, spremenim imenik} znotraj programa ali skripta ukazne lupine in dosežem, da bo to upoštevala moja trenutna ukazna lupina?

```
Subject: How do I ... and have that change affect my current shell?
Date: Thu Mar 18 17:16:55 EST 1993
```

V splošnem tega ne morete storiti, vsaj ne brez posebnih priprav. Ko nek proces ustvari svoj podproces, ta podeduje kopijo spremenljivk svojega očeta (vključno s trenutnim imenikom). Otrok lahko spremeni podedovane vrednosti, kolikor hoče, vendar te spremembe ne bodo prizadele starševskega okolja, saj otrok spreminja le svojo kopijo originalnih podatkov.

Možne so nekatere posebne priprave. Vaš otroški proces lahko izpiše spremenjene spremenljivke, če je oče vnaprej pripravljen prebrati ta izhod in ga razumeti tako, da ustrezno nastavi svoje lastne spremenljivke.

Ukazne lupine lahko izvajajo skripte tudi v okolju osnovnega procesa, namesto v okolju podprocesa, in tedaj bodo spremembe prizadele tudi osnovno ukazno lupino.

Na primer, če je vašemu skriptu za `csh` ime „`mojskript`" in ta vsebuje:

```
cd /zelo/dolga/pot setenv PATH /nekaj:/nekaj-drugega
```

ali če imate podoben skript za Bournovo ali Kornovo ukazno lupino

```
cd /zelo/dolga/pot PATH=/nekaj:/nekaj-drugega export PATH
```

in poskušate pognati „`mojskript`" v vaši ukazni lupini, se bo vaša ukazna lupina razvejila in pognala lupinski skript kot svoj podproces, v katerem bo izvedla ukaz „`cd`". Ta bo spremenil **svoj** trenutni imenik, in z ukazom „`setenv`" spremenil **svoje** okolje, vendar nobeden od ukazov ne bo vplival na trenutni imenik ukazne lupine, v katerem tipkate (vaše prijavne ukazne lupine, denimo).

Če želite prepričati vašo prijavno ukazno lupino, da požene skript v trenutnem procesu (brez razvejevanja), morate uporabiti ukaz „`.`" (za Bournovo ali Kornovo lupino) ali ukaz „`source`" (za lupino C). Se pravi, v Bournovih ali Kornovih ukaznih lupinah napišete

```
. mojaskripta
```

v lupini C pa

```
source myscript
```

Če je vse, kar poskušate storiti, le sprememba imenika ali nastavitev spremenljivke okolja, bo verjetno najlažje, da uporabite vzdevek (angl. alias) v lupini C ali funkcijo v Bournovi/Kornovi ukazni lupini. Glejte razdelek 3.4 („Kako lahko dobim trenutni imenik v moj pozivnik¿') za nekaj primerov.

Veliko bolj podroben odgovor je pripravil Thomas Michanek <`xtm@telelogic.se`>. Dobite ga lahko po FTP-ju z naslova <`ftp://ftp.wg.omron.co.jp/pub/unix-faq/docs/script-vs-env`>.

3.9 Kako lahko v `csh` ločeno preusmerim stdout in stderr?

```
Subject: How do I redirect stdout and stderr separately in csh?
From: msb@sq.com (Mark Brader)
Date: Mon, 26 Oct 1992 20:15:00 -0500
```

V `csh` lahko preusmerite stdout z „>" ter hkrati stdout in stderr z „>`&`", a ni neposrednega načina le za preusmeritev stderr. Najboljše, kar lahko naredite, je

```
( ukaz >datoteka_za_stdout ) >&datoteka_za_stderr
```

ki požene „`ukaz`" v ukazni podlupini; stdout se preusmeri znotraj podlupine v datoteko „`datoteka_za_stdout`", stdout in stderr podlupine pa se skupaj preusmerita v datoteko „`datoteka_za_stderr`", toda na tej točki je stdout že preusmerjen, tako da v datoteki „`datoteka_za_stderr`" konča pravzaprav le stderr.

Če želite le preprečiti preusmeritev stdout, tedaj to za vas stori `sh`:

```
sh -c 'ukaz 2>datoteka_za_stderr'
```

3.10 Kako znotraj .cshrc ugotovim, ali sem prijavna ukazna lupina?

```
Subject: How do I tell inside .cshrc if I'm a login shell?
Date: Thu Mar 18 17:16:55 EST 1993
```

Ko ljudje govorijo tako, pravzaprav s tem mislijo nekaj od naslednjega:

- Kako lahko ugotovim, ali gre za interaktivno ukazno lupino? ali pa
- Kako lahko ugotovim, ali gre za najvišje nivojsko ukazno lupino?

Morda lahko ugotovite, ali je vaša ukazna lupina resnično prijavna lupina (se pravi, bo prebrala „.login“ potem, ko bo opravila z „.cshrc“) tako, da zapravljate čas s „ps“ in „$$“. Prijavne lupine imajo v splošnem imena, ki se začnejo s ‚-‘. Če vas v resnici zanimata drugi dve vprašanji, je tukaj način organizacije vaše datoteke .cshrc, da to izveste.

```
if (! $?CSHLEVEL) then
        #
        # To je "najvišjenivojska" ukazna lupina,
        # morda prijavna lupina, morda lupina, pognana z
        # 'rsh stroj nek-ukaz'
        # Tukaj moramo nastaviti pot PATH in vse drugo,
        # kar želimo imeti v vsaki od naših ukaznih lupin.
        #
        setenv      CSHLEVEL        0
        set home = ~username        # za vsak slučaj
        source ~/.env               # nastavimo okolje
else
        #
        # Ta lupina je otrok ene od drugih ukaznih lupin, zato
        # nam ni treba ponovno nastaviti vseh spremenljivk okolja.
        #
        set tmp = $CSHLEVEL
        @ tmp++
        setenv      CSHLEVEL        $tmp
endif

# Izhod iz .cshrc, če ni interaktivna, npr. pod rsh.
if (! $?prompt) exit

# Tukaj smemo nastaviti pozivnik ali vzdevke, ki bi bili
# uporabni le za interaktivne ukazne lupine.

source ~/.aliases
```

3.11 Kako v lupini naredim ujemalni vzorec, ki se ujema z vsemi datotekami, razen z „.“ in „..“?

```
Subject: How do I construct a ... matches all files except "." and ".." ?
Date: Thu Mar 18 17:16:55 EST 1993
```

In vi ste mislili, da bo to preprosto ...

*****

Ustreza vsem datotekam, ki se ne začenjajo z „`.`“;

.*

Ustreza vsem datotekam, ki se začenjajo z „`.`“, toda to vključuje posebna vnosa „`.`“ in „`..`“, ki ju pogosto ne želite;

.[!.]*

(Le novejše ukazne lupine; nekatere lupine uporabljajo „`^`“ namesto „`!`“; ukazne lupine POSIX-a morajo sprejeti „`!`“, a lahko sprejmejo tudi „`^`“; prenosljive aplikacije ne bi smele uporabljati necitiranega „`^`“ takoj za „`[`“.)

Ustreza vsem datotekam, ki se začnejo z „`.`“ in jim sledi ne-„`.`“; žal bo to spregledalo „`..foo`“;

.??*

Ustreza datotekam, ki se začnejo z „`.`“ in so dolge vsaj tri znake. To se prikupno izogne „`.`“ in „`..`“, a tudi spregleda „`.a`“.

Če želite varno ujemanje z vsemi datotekami, razen „`.`“ in „`..`“, morate uporabiti 3 vzorce (če nimate imen datotek, kot je „`.a`“, lahko izpustite prvega):

```
.[!.]* .??* *
```

Alternativno lahko zaposlite zunanji program ali dva in uporabite substitucijo znotraj narekovaja (angl. backquote substitution). To (ali ‚`ls -A`‘ na nekaterih različicah Unixa) je precej dobro:

```
`ls -a | sed -e '/^\.$/d' -e '/^\.\.$/d'`
```

a tudi to bo zamočilo pri datotekah z znaki za novo vrsto, znaki IFS ali jokerjih v njihovih imenih.

V `ksh` lahko uporabite:

```
.!(.|) *
```

3.12 Kako poiščem zadnji argument v skriptu Bournove ukazne lupine?

```
Subject: How do I find the last argument in a Bourne shell script?
Date: Thu Mar 18 17:16:55 EST 1993
```

Odgovor sta prispevala:

- Martin Weitzel <`@mikros.systemware.de:martin@mwtech.uucp`>
- Maarten Litmaath <`maart@nat.vu.nl`>

Če ste prepričani, da argumentov ni več kot devet, potem lahko uporabite:

```
eval zadnji=\${$#}
```

V ukaznih lupinah, združljivih s standardom POSIX, to deluje za **vsako** število argumentov. Vselej deluje tudi:

```
for zadnji
do
        :
done
```

Primer se posploši takole:

```
for i
do
        predpredzadnji=$predzadnji
        predzadnji=$zadnji
        zadnji=$i
done
```

Zdaj pa denimo, da želite **odstraniti** zadnji argument iz seznama ali **obrniti** seznam argumentov ali za vsak *N* **dostopati** neposredno do *N*-tega argumenta. Tukaj navajamo osnovni primer, kako to storiti, in uporabimo le vgrajene konstrukte ukazne lupine, ne da bi ustvarjali dodatne podprocese:

```
t0= u0= ostanek='1 2 3 4 5 6 7 8 9' argv=

for h in '' $ostanek
do
        for t in "$t0" $ostanek
        do
                for u in $u0 $ostanek
                do
                        case $# in
                        0)
                                break 3
                        esac
                        eval argv$h$t$u=\$1
                        argv="$argv \"\$argv$h$t$u\""   # (1)
                        shift
                done
                u0=0
        done
        t0=0
done

# zdaj obnovimo argumente
eval set x "$argv"                                      # (2)
shift
```

Ta primer deluje za prvih 999 argumentov. Dovolj? Dobro poglejte vrstici, označeni z (1) in (2) in se prepričajte, da se originalni argumenti res obnovijo, ne glede na to, katere čudne znake vsebujejo!

N-ti argument lahko zdaj poiščete takole:

```
eval argN=\$argv$N
```

Če želite obrniti argumente, morate spremeniti vrstico, označeno z (1),

```
argv="\"\$argv$h$t$u\" $argv"
```

Kako odstranite zadnji argument, vam prepuščamo za nalogo.

Če dovoljujete tudi podprocese, ki morda izvajajo nevgrajene ukaze, lahko spremenljivke ‚`argvN`' določite preprosteje:

```
N=1

for i
do
        eval argv$N=\$i
        N=`expr $N + 1`
done
```

Za obračanje argumentov obstaja še preprostejša metoda, ki niti ne ustvarja podprocesov. Ta pristop lahko uporabite tudi, ko želite pobrisati npr. zadnji argument, a v tem primeru se ne morete več sklicevati neposredno na *N*-ti argument, saj so spremenljivke ‚`argv`*N*‘ nastavljene v obratnem vrstnem redu:

```
argv=

for i
do
        eval argv$#=\$i
        argv="\"\$argv$#\" $argv"
        shift
done

eval set x "$argv"
shift
```

3.13 Kaj je narobe, če imam ‚.‘ v svoji poti `$PATH`?

```
Subject: What's wrong with having '.' in your $PATH ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Malce ozadja: spremenljivka okolja `PATH` je seznam imenikov, ločenih z dvopičji. Ko vpišete ukaz brez jasno določene poti (npr. napišete „`ls`“ in ne „`/bin/ls`“), vaša ukazna lupina išče izvedljivo datoteko s tem imenom po vseh imenikih, naštetih v seznamu `PATH`, lepo po vrsti, in izvede prvi program, ki ga najde.

Eden od imenikov v seznamu `PATH` je lahko tudi trenutni imenik „`.`“. Kot znak za trenutni imenik v seznamu `PATH` lahko uporabite tudi prazni imenik. Ta dva načina označevanja sta ekvivalentna:

- za uporabnike `csh`:

  ```
  setenv PATH :/usr/ucb:/bin:/usr/bin
  setenv PATH .:/usr/ucb:/bin:/usr/bin
  ```

- za uporabnike `sh` ali `ksh`:

  ```
  PATH=:/usr/ucb:/bin:/usr/bin export PATH
  PATH=.:/usr/ucb:/bin:/usr/bin export PATH
  ```

Posestvovanje „`.`“ nekje v `PATH` je udobno - za poganjanje programov v trenutnem imeniku lahko namesto „`./a.out`“ napišete „`a.out`“. A obstaja kavelj.

Premislite, kaj se zgodi v primeru, da je „`.`“ prvi vnos v seznamu `PATH`. Denimo, da je vaš trenutni imenik javno dostopen za pisanje, na primer „`/tmp`“. Če je tam slučajno program, imenovan „`/tmp/ls`“, ki ga je tam pustil kak drug uporabnik, ter vi napišete „`ls`“ (namesto, da bi, seveda, pognali običajen program „`/bin/ls`“), bo vaša ukazna lupina pognala „`./ls`“, program tistega drugega uporabnika. Odveč je napisati, da vas lahko poganjanje neznanega programa, kot je ta, nadvse preseneti.

Bolje je postaviti oznako za trenutni imenik „`.`“ na konec `PATH`:

```
setenv PATH /usr/ucb:/bin:/usr/bin:.
```

Če ste zdaj v imeniku `/tmp` in napišete „`ls`“, bo ukazna lupina najprej preiskala `/usr/ucb`, `/bin` in `/usr/bin` za program, imenovan „`ls`“, preden bo pogledala v „`.`“, kar daje manj možnosti, da bi nehote pognali program „`ls`“ drugega uporabnika. Tudi to ni povsem varno - če ste neroden strojepisec in nekega dne napišete „`sl -l`“ namesto „`ls -l`“, tvegate, da boste pognali „`./sl`“, če ta obstaja. Kak „bister“ programer lahko računa na pogoste tipkarske napake in pušča programe s takimi imeni raztrosene po javnih imenikih. Pazite!

Veliko izkušenih uporabnikov Unixa shaja povsem dobro, ne da bi sploh imeli „`.`“ vključen v `PATH`:

```
setenv PATH /usr/ucb:/bin:/usr/bin
```

Če naredite to, boste morali za zagon programov vselej pisati „`./program`“ namesto `program`, a povečanje varnosti je verjetno vredno tega truda.

3.14 Kako lahko iz skripta ukazne lupine zapiskam na terminalu?

```
Subject: How do I ring the terminal bell during a shell script?
From: uwe@mpi-sb.mpg.de (Uwe Waldmann)
Date: Fri, 30 Apr 93 16:33:00 +0200
```

Odgovor je odvisen od različice vašega Unixa (ali raje od vrste programa „`echo`“, ki je na voljo na vašem stroju).

BSD-jevski „`echo`“ uporablja izbiro „`-n`“ za preprečitev izpisa končnega znaka za konec vrstice (angl. final newline) in ne razume osmiškega zapisa *nnn*. Ukaz se torej glasi

```
echo -n '^G'
```

kjer `^G` pomeni **dobesedni** znak BEL (vstavite ga lahko v Emacsu s kombinacijo „Ctrl-Q Ctrl-G“ ali v `vi` z uporabo „Ctrl-V Ctrl-G“).

Na Unixih SysV ukaz „`echo`“ razume zapis *nnn* in uporablja `\c` za preprečitev izpisa končnega znaka za konec vrstice, torej je odgovor:

```
echo '\007\c'
```

3.15 Zakaj ne morem uporabiti programa „`talk`“ za pogovor s prijateljem na stroju X?

```
Subject: Why can't I use "talk" to talk with my friend on machine X?
From: tmatimar@isgtec.com (Ted Timar)
Date: Thu Mar 18 17:16:55 EST 1993
```

Unix ima tri razširjene programe „`talk`“ in nobeden od njih se ne zna pogovoriti z ostalima. „Stari“ `talk` vključuje prva dva tipa. Ta različica (pogosto imenovana `otalk`) pri pogovoru med stroji ni upoštevala vrstnega reda bytov. Zaradi tega se različica `otalk` za Vax ni mogla pogovarjati z različico `otalk` za Sun. Te različice `talk`-a uporabljajo vrata 517.

Okoli leta 1987 se je večina proizvajalcev (razen Suna, ki je potreboval šest let več od svojih tekmecev) dogovorila o standardu za novi program `talk` (pogosto imenovan `ntalk`), ki ve za omrežni vrstni red bytov. Ta `talk` deluje med vsemi stroji, ki ga podpirajo. Ta različica `talk`-a uporablja vrata 518.

Dandanes obstaja nekaj programov `talk`, ki hkrati govorijo `ntalk` in eno od različic `otalk`. Najbolj znan med njimi se imenuje „`ytalk`“.

3.16 Zakaj da koledar napačen izpis?

```
Subject: Why does calendar produce the wrong output?
From: tmatimar@isgtec.com (Ted Timar)
Date: Thu Sep  8 09:45:46 EDT 1994
```

Pogosto ljudje ugotovijo, da izhod Unixovega koledarskega programa „`cal`“ ne ustreza njihovim pričakovanjem.

Koledar za september 1752 je zelo čuden:

```
   September 1752
 S  M Tu  W Th  F  S
       1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

To je mesec, v katerem so ZDA (pravzaprav celoten Britanski Imperij) zamenjale julijanski koledar z gregorijanskim.

Drug pogost problem, ki ga imajo ljudje s koledarskim programom, je, da mu podajajo argumente kot je „`cal 9 94`“. To izpiše koledar za september leta 94, **ne** pa leta 1994.

4 Vmesna vprašanja

4.1 Kako lahko ugotovim čas, ko je bila datoteka ustvarjena?

```
Date: Thu Mar 18 17:16:55 EST 1993
```

Ne morete ga - nikjer se ne shrani. Datoteke imajo podatke o času zadnje spremembe (pokaže ga „`ls -l`“), času zadnjega dostopa (pokaže ga „`ls -lu`“) in času zadnje spremembe inode (pokaže ga „`ls -lc`“). Zadnji se pogosto imenuje „čas ustvarjenja“ - celo v nekaterih straneh referenčnega priročnika (za `man`) - a to je narobe; spremeni se tudi z operacijami kot so `mv`, `ln`, `chmod`, `chown` in `chgrp`.

Pojasnila so dostopna v poglavju referenčnega priročnika o `stat(2)`.

4.2 Kako lahko uporabim `rsh`, ne da bi rsh počakal konec oddaljenega ukaza?

```
Subject: How do I use "rsh" without having the rsh hang around ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

(Glejte odgovor na vprašanje 3.7 („Zakaj dobim ob ukazu rsh gostitelj ukaz {kakšno čudno sporočilo o napaki}¿‘), o katerem „`rsh`“ govorimo.)

Očitni odgovori odpovejo:

```
rsh stroj ukaz &
```

ali

```
rsh stroj 'ukaz &'
```

Na primer, poskusite narediti

```
rsh stroj 'sleep 60 &'
```

in videli boste, da `rsh` ne bo takoj vrnil ukazne vrstice. Počakal bo 60 sekund, dokler se ne bo oddaljeni ukaz „`sleep`" izvedel do konca, čeprav se ukaz na oddaljenem stroju požene v ozadju. Kako torej pripravite `rsh` do tega, da konča z delom takoj, ko začne teči `sleep`?

Rešitev - če uporabljate na oddaljenem stroju `csh`:

```
rsh machine -n 'command >&/dev/null </dev/null &'
```

Če na oddaljenem stroju uporabljate `sh`:

```
rsh machine -n 'command >/dev/null 2>&1 </dev/null &'
```

Zakaj? „`-n`" poveže standardni vhod ukaza `rsh` na `/dev/null`, torej lahko poženete celotni ukaz `rsh` v ozadju na **lokalnem** stroju. Torej ima „`-n`" ekvivalenten pomen, kot da bi še enkrat določili „`< /dev/null`". Še več, preusmeritve vhoda in izhoda na **oddaljenem** stroju (znotraj enojnih narekovajev) zagotavljajo, da `rsh` misli, da se seja lahko prekine (ker ni več toka podatkov).

Opomba: Datoteko, na katero ali s katere preusmerjate na oddaljenem računalniku ni nujno `/dev/null`; katerakoli datoteka bo v redu.

V veliko primerih različni deli tega zapletenega ukaza niso potrebni.

4.3 Kako lahko skrajšam datoteko?

```
Subject: How do I truncate a file?
Date: Mon, 27 Mar 1995 18:09:10 -0500
```

Funkcija BSD-ja `ftruncate()` nastavi dolžino datoteke. (Toda vse različice se ne obnašajo enako.) Kaže, da tudi druge različice Unixa podpirajo neke vrste krajšanje.

Obstajajo tri znana obnašanja funkcije `ftruncate` v sistemih, ki jo podpirajo:

- BSD 4.2
  - Ultrix, SGI, LynxOS
  - krajšanje ne poveča datoteke
  - krajšanje ne premika datotečnega kazalca
- BSD 4.3
  - SunOS, Solaris, OSF/1, HP/UX, Amiga
  - krajšanje lahko poveča datoteko
  - krajšanje ne premika datotečnega kazalca
- Cray
  - UniCOS 7, UniCOS 8
  - krajšanje ne poveča datoteke
  - krajšanje spremeni datotečni kazalec

Ostali sistemi pridejo v štirih možnostih:

- `F_CHSIZE`

  - le SCO
  - nekateri sistemi definirajo izraz `F_CHSIZE`, a ga ne podpirajo
  - obnaša se kot BSD 4.3
- `F_FREESP`
  - le Interative Unix
  - nekateri sistemi (npr. Interative Unix) definirajo izraz `F_FREESP`, a ga ne podpirajo
  - obnaša se kot BSD 4.3
- `chsize()`
  - QNX in SCO
  - nekateri sistemi (npr. Interative Unix) imajo funkcijo `chsize()`, a je ne podpirajo
  - obnaša se kot BSD 4.3
- nič
  - ni znanih sistemov
  - tukaj bodo sistemi, ki sploh ne podpirajo `truncate`

Moderatorjevo sporočilo: Spodnje funkcije sem posnel nekaj let nazaj. Ne morem več ugotoviti prvotnega pisca. S. Spencer Sun <`spencer@ncd.com`> je tudi prispeval verzijo za `F_FREESP`.

Sledijo funkcije za neavtohtono funkcijo `ftruncate`.

```
/* Emulacije ftruncate(), ki delujejo na nekaterih Systemih V.
   Ta datoteka je v javni lasti. */

#include
#include

#ifdef F_CHSIZE

int
ftruncate (fd, length)
     int fd;
     off_t length;
{
  return fcntl (fd, F_CHSIZE, length);
}
#else
#ifdef F_FREESP
/* Naslednjo funkcijo je napisal
   kucharsk@Solbourne.com (William Kucharski) */

#include
#include
#include

int
ftruncate (fd, length)
```

```
     int fd;
     off_t length;
{
  struct flock fl;
  struct stat filebuf;

  if (fstat (fd, &filebuf) < 0)
    return -1;

  if (filebuf.st_size < length)
    {
      /* Extend file length. */
      if (lseek (fd, (length - 1), SEEK_SET) < 0)
        return -1;

      /* Write a "0" byte. */
      if (write (fd, "", 1) != 1)
        return -1;
    }
  else
    {
      /* Truncate length. */
      fl.l_whence = 0;
      fl.l_len = 0;
      fl.l_start = length;
      fl.l_type = F_WRLCK;      /* Zapiši ključavnico na prostor datoteke. */

      /* To se zanaša na NEDOKUMENTIRAN argument F_FREESP funkcije
         fcntl, ki skrajša datoteko tako, da se konča na položaju,
         ki ga določa fl.l_start.
         Se manjša čudesa nikoli ne bodo končala? */
      if (fcntl (fd, F_FREESP, &fl) < 0)
        return -1;
    }

  return 0;
}
#else
int
ftruncate (fd, length)
     int fd;
     off_t length;
{
  return chsize (fd, length);
}
#endif
#endif
```

4.4 Zakaj simbol „{}“ ukaza `find` ne naredi to, kar želim?

```
Subject: Why doesn't find's "{}" symbol do what I want?
Date: Thu Mar 18 17:16:55 EST 1993
```

Ukaz `find` ima izbiro `-exec`, ki izvede podani ukaz na vseh izbranih datotekah. Ukaz `find` zamenja vse nize „{}“, ki jih vidi v ukazni vrstici, z imenom datoteke, ki jo trenutno obdeluje.

Tako lahko lepega dne poskusiti uporabiti `find`, da bi pognali ukaz na vsaki datoteki, imenik po imenik. Poskusite lahko tole:

```
find /pot -type d -exec ukaz {}/\* \;
```

upajoč, da bo `find` izvajal ukaze takole:

```
ukaz imenik1/*
ukaz imenik2/*
...
```

Žal `find` razvije simbol „{}“ le, ko nastopa sam. Vse drugo, kot na primer „`{}/*`“ bo `find` pustil pri miru, torej bo namesto želenega dejanja izvedel

```
ukaz {}/*
ukaz {}/*
...
```

enkrat za vsak imenik. To je lahko hrošč, lahko je odlika, a shajati moramo s takim vedenjem.

Kako se mu izognemo? En način je s trivialnim kratkim skriptom v ukazni lupini, denimo „`./naredi`“, ki vsebuje le

```
ukaz "$1"/*
```

Potem lahko uporabljate

```
find /pot -type d -exec ./naredi {} \;
```

Ali, če se želite izogniti skriptu „`./naredi`“, lahko uporabite

```
find /pot -type d -exec sh -c 'ukaz $0/*' {} \;
```

(To deluje, ker se znotraj ‚ukaza‘ „`sh -c 'ukaz' A B C ...`“, `$0` razvije v `A`, `$1` v `B`, in tako naprej.)

Ali pa uporabite trik sestavljanja ukaza s pripomočkom `sed`:

```
find /pot -type d -print | sed 's:.*:ukaz &/*:' | sh
```

Če na vsak način želite zmanjšati število klicev ukaza „`ukaz`“, lahko pogledate, če ima vaš sistem ukaz `xargs`. Ukaz `xargs` prebere argumente po vrsticah s standardnega vhoda in jih zbere, kolikor le lahko, v eno samo ukazno vrstico. Uporabite lahko torej

```
find /pot -print | xargs ukaz
```

kar bo pognalo enkrat ali večkrat

```
ukaz datoteka1 datoteka2 datoteka3 imenik1/datoteka1 imenik1/datoteka2
```

Žal to ni popolnoma robustna ali varna rešitev. `Xargs` pričakuje, da se vhodne vrstice končajo z znakom za novo vrsto (angl. newline), zato se bo zmedel ob datotekah, poimenovanih s čudnimi znaki, kakršni so znaki za novo vrsto.

4.5 Kako lahko zaščitim simbolične povezave?

```
Subject: How do I set the permissions on a symbolic link?
Date: Thu Mar 18 17:16:55 EST 1993
```

Zaščite (angl. permissions, dovolilnice tipa `-rw-r-r-`) pri simboličnih povezavah pravzaprav ne pomenijo ničesar. Edina dovoljenja, ki štejejo, so zaščite datotek, na katere kažejo povezave.

4.6 Kako lahko „obnovim" pobrisano datoteko?

```
Subject: How do I "undelete" a file?
Date: Thu Mar 18 17:16:55 EST 1993
```

Nekoč boste po pomoti napisali nekaj kot „`rm * .foo`" in ugotovili, da ste pravkar pobrisali „`*`" namesto „`*.foo`". Mislite si, da je to obred minljivosti.

Seveda bi moral vsak spodoben upravitelj sistema redno delati varnostne kopije. Preverite pri svojem upravitelju, če je na voljo nedavna varnostna kopija vaše datoteke. Če te ni, berite naprej.

Pri vseh okoliščinah ko enkrat pobrišete datoteko z ukazom „`rm`", te ni več. Ko na datoteki uporabite ukaz „`rm`", sistem popolnoma pozabi, kateri bloki, razsejani po disku, so bili del vaše datoteke. Še huje - bloki pravkar pobrisane datoteke bodo prvi uporabljeni in prepisani, ko bo sistem potreboval več diskovnega prostora. Vendar nikoli ne recite nikoli. Teoretično je mogoče vrniti delčke podatkov, **če** takoj po ukazu `rm` zaustavite sistem (angl. shut down). A tedaj raje imejte pri roki zelo čarovniški tip osebe, ki ima na voljo dolge ure ali dneve, da vam obnovi pobrisano.

Ko po pomoti pobrišete datoteko z `rm`, bo prvi nasvet, da naredite vzdevek v ukazni lupini ali pa proceduro, ki bo spremenila ukaz „`rm`" tako, da boste datoteko namesto pobrisali, raje prestavili v koš za smeti. Tako lahko datoteke obnovite, če napravite napako, a morate občasno prazniti koš za smeti. Dve opombi: prvič, splošno mnenje je, da je to **slaba** ideja. Na novo lastnost ukaza se boste vselej zanašali in ko se boste znašli na običajnem sistemu, kjer ukaz „`rm`" zares izvede „`rm`", boste prišli v težave. Drugič, verjetno boste ugotovili, da je ob vsej zmešnjavi z diskovnim prostorom in časom, porabljenim za vzdrževanje koša za smeti, vseeno lažje malo bolj paziti pri uporabi ukaza „`rm`". Za začetnike zadošča, da si v vašem priročniku ogledate izbiro „`-i`" ukaza „`rm`".

Če ste še vedno pogumni, potem je tukaj možen preprost odgovor. Naredite si ukaz „`can`", ki bo premikal datoteke v imenik-smetnjak `trashcan`. V `csh(1)` lahko postavite naslednje ukaze v datoteko „`.login`" v vašem domačem imeniku:

```
alias can       'mv \!* ~/.trashcan'       # vrzi datoteko/e v koš
alias mtcan     'rm -f ~/.trashcan/*'      # nepovratno izprazni koš
if ( ! -d ~/.trashcan ) mkdir ~/.trashcan  # zagotovi obstoj koša
```

Morda boste tudi želeli postaviti:

```
rm -f ~/.trashcan/*
```

v datoteko „`.logout`" v vašem domačem imeniku, da boste samodejno izpraznili koš vselej, ko se odjavite. (Različici za ukazni lupini `sh` in `ksh` sta prepuščeni bralcu kot vaja.)

Pri projektu Athena na MIT-u je nastal obsežen paket za brisanje/vrnitev/izbris/čiščenje, ki lahko služi kot popolno nadomestilo za `rm`, vendar dopušča obnovitev datotek. Ta paket je bil objavljen v `comp.sources.misc` (volume 17, issue 023-026)

4.7 Kako lahko proces ugotovi, ali teče v ozadju?

```
Subject: How can a process detect if it's running in the background?
Date: Thu Mar 18 17:16:55 EST 1993
```

Najprej razčistimo: želite vedeti, ali tečete v ozadju ali želite vedeti, ali tečete interaktivno? Če se odločate, ali boste izpisovali pozivnik in podobno, je to verjetno boljši kriterij. Poglejte, če je standardni vhod terminal:

```
sh:  if [ -t 0 ]; then ... fi
C:  if(isatty(0)) { ... }
```

V splošnem ne morete preveriti, ali tečete v ozadju. Osnovni problem je, da imajo različne ukazne lupine in različne različice Unixa različne predstave o tem, kaj pomenita „ospredje“ in „ozadje“ - in na najbolj pogostem tipu sistema z dobro razčiščenima pojmoma se lahko programi poljubno premikajo med ospredjem in ozadjem!

Sistemi Unix brez nadzora opravil tipično postavijo proces v ozadje tako, da ignorirajo `SIGINT` in `SIGQUIT` ter preusmerijo standardni vhod na „`/dev/null`“; to naredi ukazna lupina.

Ukazne lupine, ki podpirajo nadzor opravil (angl. job control), na sistemih Unix, ki podpirajo nadzor opravil, postavijo proces v ozadje tako, da mu priredijo identifikacijsko številko ID različno od skupine procesov, ki pripadajo terminalu. V ospredje ga premaknejo tako, da popravijo skupinsko procesno številko terminala (angl. terminal's process group ID) s procesno številko. Ukazne lupine, ki **ne** podpirajo nadzora opravil, na sistemih Unix, ki podpirajo nadzor opravil, tipično naredijo isto kot ukazne lupine na sistemih, ki ne podpirajo nadzora opravil.

4.8 Zakaj preusmerjanje v zanki ne dela, kot bi moralo? (Bournova lupina)

```
Subject: Why doesn't redirecting a loop work as intended?  (Bourne shell)
Date: Thu Mar 18 17:16:55 EST 1993
```

Oglejte si naslednji primer:

```
foo=bar

while read line
do
        # naredi nekaj z $line
        foo=bletch
done < /etc/passwd

echo "foo je zdaj: $foo"
```

Kljub prireditvi „`foo=bletch`“ bo to izpisalo „`foo je zdaj: bar`“ v mnogih izvedbah Bournove ukazne lupine. Zakaj? Zaradi naslednje, pogosto nedokumentirane lastnosti zgodovinskih Bournovih ukaznih lupin: preusmerjanje kontrolne strukture (kot je zanka ali stavek „`if`“) naredi ukazno podlupino, v kateri se struktura izvede; spremenljivke, nastavljene v podlupini (kot priredba „`foo=bletch`“), seveda ne vplivajo na trenutno lupino.

Odbor za standardizacijo ukaznih lupin in vmesnika orodij POSIX 1003.2 prepoveduje zgornje neutemeljeno ravnanje, se pravi, na sistemu, usklajenemu s P1003.2, v Bournovih ukaznih lupinah zgornji primer izpiše „`foo je zdaj: bletch`“.

V zgodovinskih (in s P1003.2 usklajenih) izvedbah uporabite naslednji trik, da se izognete težavam s preusmerjanjem:

```
foo=bar
```

```
# datotečni deskriptor 9 naj postane duplikat datotečnega deskriptorja 0 (stdin);
# potem poveži stdin s /etc/passwd; originalni stdin smo si zdaj
# ‚zapomnili‘ v datotečnem deskriptorju 9; glej dup(2) in sh(1)
exec 9<&0 < /etc/passwd

while read line
do
        # naredi nekaj z $line
        foo=bletch
done

# naj bo stdin duplikat datotečnega deskriptorja 9, se pravi,
# spet ga povežemo z originalnim stdin; zatem zapremo deskriptor 9
exec 0<&9 9<&-

echo "foo je zdaj: $foo"
```

To bi moralo vselej izpisati „`foo je zdaj; bletch`“. Prav, vzemite naslednji primer:

```
foo=bar

echo bletch | read foo

echo "foo je zdaj: $foo"
```

To bo izpisalo „`foo je zdaj: bar`“ v veliko izvedbah in „`foo je zdaj: bletch`“ v nekaterih drugih. Zakaj? V splošnem se vsak del cevovoda izvede v drugi ukazni podlupini; v nekaterih izvedbah pa je zadnji ukaz v cevovodu izjema: če je to vgrajeni ukaz, kakršen je „`read`“, ga bo izvedla trenutna lupina, sicer pa bo izveden v podlupini.

POSIX 1003.2 dovoljuje oba načina, zaradi česar se prenosljivi skripti ne morejo zanašati na nobenega od njiju.

4.9 Kako poženem `passwd`, `ftp`, `telnet`, `tip` in druge interaktivne programe v skriptu ukazne lupine v ozadju?

```
Subject: How do I run ... interactive programs from a shell script ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Ti programi pričakujejo terminalski vmesnik. Ukazne lupine jim ga posebej ne priskrbijo. Torej, ti programi ne morejo biti avtomatizirani v skriptih ukaznih lupin.

Program „`expect`“ poskrbi za programabilen terminalski vmesnik za avtomatsko interakcijo s takimi programi. Naslednji skript za `expect` je primer neinteraktivne različice ukaza `passwd(1)`.

```
# uporabniško ime je dano kot 1. argument, geslo kot 2.
set geslo [index $argv 2]
spawn passwd [index $argv 1]
expect "*password:"
send "$geslo\r"
expect "*password:"
send "$geslo\r"
expect eof
```

Program `expect` lahko delno avtomatizira interakcijo, kar je posebej uporabno za `telnet`, `rlogin`, razhroščevalnike in druge programe, ki nimajo vgrajenega ukaznega jezika. Distribucija priskrbi tudi poskusni skript za nenehen zagon igre `rogue`, dokler se ne pojavi dobra začetna postavitev. Potem se nadzor vrne uporabniku, ki lahko uživa v igri.

Na srečo so bili napisani nekateri programi za obdelavo povezave na terminalu pseudo-tty, tako da lahko poganjate te vrste programov v skriptu.

Program `expect` dobite tako, da pošljete „`send pub/expect/expect.shar.Z`" na naslov `library@cme.nist.gov` ali uporabite anonimni FTP za prenos imenovane datoteke s strežnika `ftp.cme.nist.gov`.

Še ena možnost rešitve je z uporabo programa pty 4.0, ki deluje kot program pod sejo psevdo-tty, in ga dobite na `comp.sources.unix`, volume 25. Postopek, ki to naredi prek `pty` z uporabo poimenovanih cevi, je videti takole:

```
#!/bin/sh
/etc/mknod out.$$ p; exec 2>&1
( exec 4<out.$$; rm -f out.$$
<&4 waitfor 'password:'
    echo "$2"
<&4 waitfor 'password:'
    echo "$2"
<&4 cat >/dev/null
) | ( pty passwd "$1" >out.$$ )
```

Tukaj je „`waitfor`" preprost C-jevski program, ki bere s standardnega vhoda znak po znak, dokler ti znaki niso enaki njegovemu argumentu.

Preprostejša rešitev z uporabo `pty` (ki ima to napako, da se ne sinhronizira pravilno s programom `passwd`) je

```
#!/bin/sh
( sleep 5; echo "$2"; sleep 5; echo "$2") | pty passwd "$1"
```

4.10 Kako lahko v skriptu ali programu ugotovim ID procesa, ki pripada programu z določenim imenom?

```
Subject: How do I find the process ID of a program with a particular name ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

V skriptu ukazne lupine:

Doslej še ni pripomočka, ki bi bil posebej načrtovan za preslikovanje med imeni programov in procesnimi ID-ji. Še več, takšne preslikave so pogosto nezanesljive, saj lahko obstaja več procesov z enakim imenom in ker je možno, da proces med tekom spreminja svoje ime. Toda pogosto zadošča, da uporabimo naslednji cevovod, ki izpiše seznam procesov (v vaši lasti) z določenim imenom:

```
ps ux | awk '/ime/ && !/awk/ {print $2}'
```

Pri tem zamenjajte „`ime`" z imenom procesa, ki ga iščete.

Splošni postopek prestreže izhod ukaza `ps` in z orodji `awk` ali `grep` in podobnimi poišče vrstice z določenim nizom znakov ter izpiše polje PID v teh vrsticah. Opazili boste, da zgornji ukaz „`!/awk/`" iz izpisa izloči sočasni proces `awk`.

Morda boste morali v odvisnosti od vrste Unixa, ki ga uporabljate, spremeniti argumente za `ps`.

V kodi programa v C-ju:

Prav tako kot ni nobenega pripomočka, posebej načrtovanega za preslikavo med imeni programov in procesnimi ID-ji, tudi ni (prenosljive) C-jevske knjižnice funkcij za to opravilo.

Nekateri proizvajalci sicer priskrbijo funkcije za branje pomnilnika jedra; na primer, Sun vključuje funkcije „`kvm_`" in Data General funkcije „`dg_`". Možno je, da jih sme uporabljati vsak uporabnik, a morebiti so dostopne le superuporabniku (ali uporabniku v skupini „`kmem`"), če je na vašem sistemu omejeno branje pomnilnika jedra. Nadalje so te funkcije pogosto nedokumentirane ali vsaj slabo dokumentirane in se lahko spreminjajo od izdaje do izdaje.

Nekateri proizvajalci naredijo datotečni sistem „/proc“, ki se kaže kot imenik s kupom datotek v njem. Datoteke so poimenovane s števili, ki ustrezajo ID procesov. Te datoteke lahko odpirate in berete ter tako dobite podatke o procesu. Kot rečeno pa je dostop do tega imenika morebiti omejen in vmesnik se lahko spreminja od sistema do sistema.

Če ne morete uporabiti posebnih proizvajalčevih knjižničnih funkcij in če nimate sistema /proc ter bi radi vse postorili v C-ju, boste morali sami brskati po pomnilniku jedra. Za dober primer, kako se to počne na večih sistemih, poglejte izvorno kodo „ofiles“, dostopno v arhivih novičarske skupine comp.sources.unix. (Paket, imenovan „kstuff“, ki pomaga pri brskanju po jedru, je bil poslan na alt.sources v maju 1991 in je tudi dostopen po anonimnem FTP-ju kot {329{6,7,8,9},330{0,1}} v imeniku usenet/alt.sources/articles/ na wuarchive.wustl.edu.

4.11 Kako preverim izhodni status oddaljenega ukaza, pognanega z **rsh**?

```
Subject: How do I check the exit status of a remote command executed via "rsh"?
Date: Thu Mar 18 17:16:55 EST 1993
```

Tole ne deluje:

```
rsh nek-stroj nek-zoprn-ukaz || echo "Ukazu je spodletelo"
```

Izhodno stanje „rsh“ je 0 (uspeh), če se sam program rsh konča uspešno, kar verjetno ni to, kar želite.

Če želite preveriti izhodni status oddaljenega programa, lahko uporabite skript „ersh“ Maartena Litmaatha, ki je bil poslan na alt.sources v oktobru 1994. Skript ersh je lupinski skript, ki pokliče rsh, uredi, da oddaljen stroj na koncu izpiše status zadnjega izvedenega ukaza, in konča v tem izhodnem stanju.

4.12 Je mogoče programu **awk** podati tudi spremenljivke ukazne lupine?

```
Subject: Is it possible to pass shell variable settings into an awk program?
Date: Thu Mar 18 17:16:55 EST 1993
```

Za to obstajata dva različna načina. S prvim preprosto razvijemo spremenljivke povsod, kjer je to potrebno v programu. Na primer, seznam vseh terminalov tty, ki jih uporabljate, dobite takole:

```
who | awk '/^'"$USER"'/ { print $2 }'
```

Enojni narekovaji se uporabljajo za označevanje programa v awk-u, saj se v njem pogosto uporablja znak ‚$‘, ki je lahko interpretiran v ukazni lupini, če zaprete program v dvojne narekovaje, ne pa tudi, če ga zaprete v enojne. V tem primeru **hočemo**, da ukazna lupina interpretura ‚$‘ v „$USER“, zato zapremo enojne narekovaje in potem vstavimo „$USER“ v dvojne. Pozorni bodite na to, da ni nikjer nobenih presledkov, torej bo ukazna lupina vse skupaj videla kot en sam argument. Opazili boste tudi, da dvojni narekovaji v tem konkretnem primeru verjetno niso potrebni, lahko bi torej naredili tudi

```
who | awk '/^'$USER'/ { print $2 }'
```

a jih moramo vseeno vključiti, saj bi lahko lupinska spremenljivka $USER vsebovala posebne znake, kot so presledku.

Drugi način za podajanje spremenljivih nastavitev programu awk je uporaba pogosto nedokumentirane lastnosti awk-a, ki dovoljuje, da se spremenljivke podajajo v ukazni vrstici kot „lažna imena datotek“. Na primer:

```
who | awk '$1 == uporabnik { print $2 }' uporabnik="$USER" -
```

Spremenljivke se uporabijo, ko se zasledijo v ukazni vrstici, torej, na primer, lahko s to tehniko naročite programu awk, kako naj se obnaša za različne datoteke. Na primer:

```
awk '{ program, ki je odvisen od s }' s=1 datoteka1 s=0 datoteka2
```

Opazite, da nekatere verzije programa awk povzročijo, da se pred izvedbo bloka BEGIN zasledijo spremenljivke pred vsemi pravimi imeni datotek, toda nekatere ne, zato se na to ne gre zanašati.

Opazite tudi, da awk ob navedbi spremenljivk namesto pravih datotek sam od sebe ne bo bral s stdin, zato morate dodati argument „-“ na konec vašega ukaza, kot je to v predprejšnjem primeru.

Tretja možnost je uporaba novejše verzije programa awk (nawk), ki dopušča neposreden dostop do spremenljivk okolja. Na primer:

```
nawk 'END { print "Vaša pot je " ENVIRON["PATH"] }' /dev/null
```

4.13 Kako se znebim procesov-zombijev, ki vztrajajo?

```
Subject: How do I get rid of zombie processes that persevere?
From: Jonathan I. Kamens
From: casper@fwi.uva.nl (Casper Dik)
Date: Thu, 09 Sep 93 16:39:58 +0200
```

Žal je nemogoče posplošiti, kako naj se zaključi podproces, saj se na različnih vrstah Unixa mehanizmi medsebojno razlikujejo.

Predvsem morate narediti wait() za podproces na **vseh** vrstah Unixa. Ne poznam Unixa, ki bi avtomatično pospravil za podprocesom, ki se je končal, če mu tega niste posebej naročili.

Drugič, če na nekaterih, iz SysV izpeljanih, sistemih naredite „signal(SIGCHLD, SIG_IGN)“ (no, pravzaprav je lahko SIGCLD namesto SIGCHLD, toda večina novejših sistemov SysV ima „#define SIGCHLD SIGCLD“ v datotekah z glavami), se otroški procesi samodejno počistijo, brez nadaljnega truda na vaši strani. Najpreprosteje to preverite na svojem stroju, tako da poskusite, ali deluje, čeprav se ni priporočljivo zanašati na to, če poskušate pisati prenosljivo kodo. Žal POSIX tega ne predpisuje. Nastavitev SIGCHLD na SIG_IGN je v POSIX-u nedefinirana, torej tega ne morete uporabiti v svojem programu, če naj bo skladen s standardom POSIX.

Kakšen je potemtakem način, skladen s POSIX? Kot rečeno, morate namestiti upravljalnik signalov (angl. signal handler) in počakati z wait. V POSIX-u so upravljalniki signalov nameščeni s funkcijo sigaction. Ker vas ne zanimajo „zaustavljeni“ podprocesi, pač pa prekinjeni, dodajte zastavicam sa_flags vrednost SA_NOCLDSTOP. Čakanje brez blokiranja se izvede z waitpid(). Prvi argument funkciji waitpid naj bo -1 (čakaj na katerikoli PID), tretji naj bo WNOHANG. To je danes najbolj prenosljiv način in bo verjetno v prihodnosti postal še bolj prenosljiv.

Če vaš sistem ne podpira standard POSIX, obstaja vrsta rešitev. Najpreprosteje je uporabiti signal(SIGCHLD, SIG_IGN), če to deluje. Če samodejnega čiščenja ne morete doseči s SIG_IGN, morate napisati upravljalnik signalov, da bo to storil. Zaradi naslednjih nekonsistentnosti ga sploh ga ni preprosto napisati tako, da bi deloval na vseh vrstah Unixa:

Na nekaterih vrstah Unixa se upravljalnik signalov SIGCHLD pokliče, če se konča eden **ali več** podprocesov. Če vaš upravljalnik signalov le enkrat pokliče wait(), to pomeni, da ne bo počistil za vsemi podprocesi. Prepričan sem, da ima na srečo programer na teh Unixih, na katerih to drži, vselej na voljo klica wait3() ali waitpid(), ki z izbiro WNOHANG preverita, ali morata počistiti še kaj podprocesov. Vaš upravljalnik signalov na sistemih, ki poznajo funkciji wait3()/waitpid(), toliko časa kliče eno od teh funkcij z izbiro WNOHANG, dokler ni več nobenih otrok za počiščenje. Boljša izbira je waitpid(), ker je vključena v POSIX.

Na sistemih, izpeljanih iz SysV, se signali SIGCHLD regenerirajo, če po izhodu upravljalnika signala SIGCHLD še vedno ostane kakšen otroški proces za počistiti. Torej je na večini sistemov SysV ob klicu upravljalnika signalov varno predpostaviti, da morate počistiti le en signal, in pričakovati, da se bo upravljalnik poklical ponovno, če je po izhodu iz njega ostalo še kaj nepočiščenih signalov.

Na starejših sistemih ni nobenega načina, da bi preprečili upravljalnikom signalov, da se ob klicu samodejno nastavijo na `SIG_DFL`. Na takih sistemih morate kot zadnjo stvar upravljalnika signalov postaviti „`signal(SIGCHILD, catcher_func)`“ (kjer je „`catcher_func`“ ime upravljalniške funkcije), da se resetirajo.

Na srečo novejše izvedbe dovoljujejo upravljalnikom signalov, da se namestijo, ne da bi bili resetirani na `SIG_DFL`, ko se požene upravljalniška funkcija. Temu problemu se na sistemih, ki nimajo `wait3()`/`waitpid()`, a imajo `SIGCLD`, izognete tako, da morate resetirati upravljalnik signalov s klicem `signal()`, ko ste izvedli v upravljalniku vsaj en `wait()`, vsakič, ko se kliče. Zaradi zgodovinskih združljivostnih razlogov bo System V obdržal staro pomenoslovje (resetiral upravljalnik ob klicu) funkcije `signal()`. Upravljalniki signalov, ki se prilepijo, se lahko namestijo s `sigaction()` ali `sigset()`.

Povzetek vsega tega je, da morate funkcijo `waitpid()` (POSIX) ali `wait3()` na sistemih, ki jo imajo, tudi uporabljati in, da mora teči vaš upravljalnik signalov v zanki, na sistemih, ki pa teh funkcij nimajo, morate imeti ob vsaki zahtevi po upravljalniku signalov po en klic `wait()`.

Še ena stvar - če ne želite iti skozi vse te težave, obstaja prenosljiv način, da se izognete temu problemu, čeprav je malce manj učinkovit. Vaš starševski proces naj se razveji (s `fork`) in potem čaka na mestu, da se konča otroški proces. Otroški proces se potem spet razveji, in vam da otroka in vnuka. Otrok se takoj prekine (in torej starš, ki ga čaka, izve za njegov konec in nadaljuje z delom), vnuk pa počne, kar bi moral prvotno narediti otrok. Ker je njegov starš končan, ga nasledi proces `init`, ki bo opravil vse potrebno čakanje. Ta metoda je neučinkovita, ker potrebuje dodatno razvejanje, a je popolnoma prenosljiva.

4.14 Kako dobim vrstice iz cevovoda tako, kot se pišejo, namesto le v večjih blokih?

```
Subject: How do I get lines from a pipe ... instead of only in larger blocks?
From: Jonathan I. Kamens
Date: Sun, 16 Feb 92 20:59:28 -0500
```

Knjižnica stdio trenutno dela vmesno pomnenje (angl. buffering) različno, odvisno od tega, ali misli, da teče na tty ali ne. Če misli, da je na tty, dela vmesno pomnenje po posameznih vrsticah; sicer uporablja večji vmesni pomnilnik kot je ena vrstica.

Če imate izvorno kodo odjemnika, za katerega bi radi onemogočili vmesno pomnenje, lahko za to uporabite `setbuf()` ali `setvbuf()`.

Če ne, je najboljši način, da prepričate program, da teče na tty tako, da ga poženete pod pty, se pravi, da uporabite program „`pty`“, omenjen v vprašanju 4.9 („Kako poženem passwd, ftp, telnet, tip in druge interaktivne programe v skriptu ukazne lupine v ozadju¿‘).

4.15 Kako lahko vstavim datum v ime datoteke?

```
Subject: How do I get the date into a filename?
From: melodie neal <melodie@comtech.ct.oz.au>
Date: Fri, 7 Oct 1994 09:27:33 -0400
```

To ni težko, a je na prvi pogled malce kriptično. Začnimo najprej s samim ukazom `date`: `date` lahko vzame niz s formatom izpisa in ga upošteva. Niz za formatiranje mora biti objet z narekovaji, da preprečimo ukazni lupini poskus njegove interpretacije, preden ga dobi sam ukaz `date`. Poskusite tole:

```
date '+%d%m%y'
```

Morali bi dobiti nekaj kot 130994. Če želite imeti vmes ločila, preprosto postavite znake, ki bi jih radi uporabljali, v niz za formatiranje (**brez nagibnic ‚/‘**):

```
date '+%d.%m.%y'
```

Obstaja veliko simbolov, ki jih lahko uporabite v nizu za formatiranje: o njih izveste v strani referenčnega priročnika (`man date`).

Zdaj pa še to spravimo v ime datoteke. Denimo, da bi radi ustvarili datoteke imenovane `report.130994` (ali katerikoli datum že je danes):

```
FILENAME=report.`date '+%d%m%y'`
```

Opazite, da tukaj uporabljamo dva nabora narekovajev: notranji nabor preprečuje formatirnemu nizu prezgodnjo interpretacijo; zunanji nabor pove ukazni lupini naj izvede objeti ukaz, in zamenja rezultat v izraz (zamenjava ukazov, angl. command substitution).

4.16 Zakaj se nekateri skripti začnejo z „#! . . .“?

```
Subject: Why do some scripts start with #! ... ?
From: chip@@chinacat.unicom.com (Chip Rosenthal)
Date: Tue, 14 Jul 1992 21:31:54 GMT
```

Chip Rosenthal je v preteklosti v novičarski skupini `comp.unix.xenix` odgovoril na zelo podobno vprašanje.

Mislim, da ljudi bega, da obstajata dva različna mehanizma, ki se oba začneta z znakom ‚`#`‘. Oba rešujeta isti problem na zelo omejenem naboru primerov - a nista nič manj različna.

Nekaj ozadja. Ko se jedro sistema UNIX odloči pognati program (eden sistemskih klicev družine `exec()`), najprej pogleda prvih 16 bitov datoteke. Teh 16 bitov se imenuje *magična številka*. Prvič, magična številka preprečuje jedru, da bi naredilo kaj neumnega, kot na primer poskušalo izvesti datoteko z bazami podatkov o vaših strankah. Če jedro ne prepozna magične številke, se pritoži z napako `ENOEXEC`. Program požene le, če je magična številka prepoznavna.

Drugič, sčasoma so se uvajali različni formati izvedljivih datotek in magična številka ni le povedala jedru `ali` naj izvede datoteko, pač pa tudi **kako** naj jo izvede. Na primer, če prevedete program na sistemu SCO XENIX/386 in nesete binarno datoteko na sistem SysV/386, bo jedro prepoznalo magično številko in reklo „Aha! To je binarna datoteka tipa x.out¡‘ in se nastavilo tako, da bo uporabljalo klice, združljive s sistemom XENIX.

Pomnite, da jedro lahko poganja le binarne izvedljive slike pomnilnika. Kako torej, se vprašujete, tečejo skripti? Konec koncev, lahko napišem v pozivniku ukazne lupine „`moj.skript`“ in ne dobim napake `ENOEXEC`. Odgovor: skripti se ne izvajajo v jedru, pač pa v ukazni lupini. Koda v ukazni lupini lahko izgleda podobno:

```
/* poskusi pognati program */
execl(program, basename(program), (char *)0);

/* klic exec je spodletel - morda gre za skript ukazne lupine? */
if (errno == ENOEXEC)
     execl ("/bin/sh", "sh", "-c", program, (char *)0);

/* Oh, ne, g. Bill!! */
perror(program);
return -1;
```

(Ta primer je močno poenostavljen. Vpletenih je veliko več zadev, a to ponazarja bistvo, ki ga poskušam opisati.)

Če je klic `execl()` uspešen in se program zažene, se koda za `execl()` nikoli ne požene. V zgornjem primeru uspešen `execl()` programa ‚`program`‘ pomeni, da se spodnje vrstice primera ne izvedejo. Namesto tega sistem izvaja binarno datoteko ‚`program`‘.

Če, po drugi strani, prvi klic `execl()` spodleti, ta hipotetična ukazna lupina pogleda, zakaj ji je spodletelo. Če klic `execl()` ni uspel, ker ‚`program`' ni bil prepoznan kot binarna izvedljiva datoteka, ga ukazna lupina poskuša naložiti kot skript ukazne lupine.

Ljudje iz Berkeleya so imeli čedno idejo, kako razširiti način, na katerega jedro zaganja programe. Popravili so jedro, da prepozna magično številko ‚`#!`' (magične številke so velike 16 bitov in dva 8-bitna znaka tvorita 16 bitov, prav?). Ko je jedro prepoznalo magično številko ‚`#!`', je prebralo tudi ostanek vrstice in ga obravnavalo kot ukaz, ki ga naj požene na vsebini datoteke. S tem popravkom lahko zdaj počnete podobne stvari:

```
#! /bin/sh

#! /bin/csh

#! /bin/awk -F:
```

Ta popravek je obstajal le v svetu Berkeleya, in je prešel na jedra USG kot del sistema System V Release 4. Pred V.4 jedro ni imelo možnosti ničesar drugega, kot nalaganje in zaganjanje binarnih izvedljivih slik pomnilnika, razen v primerih proizvajalčeve dodane vrednosti.

Vrnimo se še nekaj let v preteklost, v čas, ko je več in več ljudi, ki so poganjali Unixe na jedrih USG, govorilo „`/bin/sh` je zanič kot interaktivni uporabniški vmesnik! Hočem `csh`¡'. Nekateri proizvajalci so dodali `csh` v njihove distribucije, čeprav `csh` ni bil del distribucije USG UNIX.

To pa je predstavljalo problem. Denimo, da spremenite svojo prijavno ukazno lupino na `/bin/csh`. Denimo, nadalje, da ste bebec in vztrajate na programiranju skriptov `csh`. Vsekakor bi radi bili sposobni napisati ‚`moj.skript`' in ga s tem pognati, čeprav je to skript za `csh`. Namesto, da bi ga pognali skozi `/bin/sh`, želite, da se skript požene takole:

```
execl ("/bin/csh", "csh", "-c", "moj.skript", (char *)0);
```

Kaj pa vsi tisti obstoječi skripti – nekateri od njih so deli sistemske distribucije? Če se poženejo skozi `csh`, se bodo stvari kvarile. Potrebujete torej način, da poženete nekatere skripte skozi `csh`, in nekatere druge skozi `sh`.

Vpeljana rešitev je bila, da se popravi `csh` tako, da pogleda prvi znak skripta, ki ga želite pognati. Če je ta ‚`#`', bo `csh` zagnal skript skozi `/bin/csh`, sicer bo pognal skript skozi `/bin/sh`. Primer zgornje kode bi zdaj izgledal takole:

```
/* poskusi pognati program */
execl(program, basename(program), (char *)0);

/* klic exec je spodletel - morda gre za skript ukazne lupine? */
if (errno == ENOEXEC && (fp = fopen(program, "r")) != NULL) {
i = getc(fp);
(void) fclose(fp);
if (i == '#')
    execl ("/bin/csh", "csh", "-c", program, (char *)0);
else
    execl ("/bin/sh", "sh", "-c", program, (char *)0);
}

/* Oh, ne, g. Bill!! */
perror(program);
return -1;
```

Dve pomembni stvari, Prvič, to je popravek lupine ‚`csh`'. V jedru se ni nič spremenilo in drugim ukaznim lupinam ni bilo nič dodano. Če poskušate pognati skript s klicem `execl()`, boste še vedno dobili napako `ENOEXEC`, pa naj se skript začne z znakom ‚#‘ ali pa ne. Če poskušate pognati skript, ki se začne z znakom ‚#‘ v čem drugem kot csh (na primer v /bin/sh), ga bo obravnavala lupina sh, ne csh.

Drugič, čarovnija je v tem, da se bodisi skript začne z ‚#‘, ali pa se ne začne z ‚#‘. Stvari kot so ‚:‘ in ‚: /bin/sh‘ na začetku skriptov dela čarobne preprosto dejstvo, da niso ‚#‘. Torej je vse to identično, če je na začetku skripta:

```
:

: /bin/sh

                <--- prazna vrstica

: /usr/games/rogue

echo "Hm ... sprašujem se, pod katero ukazno lupino tečem???"
```

V vseh teh primerih bodo vse ukazne lupine poskušale izvesti skript z /bin/sh.

Podobno, tudi vse naslednje je identično, če je na začetku skripta:

```
#

# /bin/csh

#! /bin/csh

#! /bin/sh

# Hm ... sprašujem se, pod katero ukazno lupino tečem???
```

Vse te vrstice se začnejo z znakom ‚#‘. To pomeni, da se bo skript pognal **le**, če ga boste poskušali pognati iz csh, sicer se bo pognal z /bin/sh.

(Opomba: če poganjate ksh, zamenjajte v zgornjem besedilu „sh“ s „ksh“. Kornova ukazna lupina je teoretično združljiva z Bournovo, torej poskuša sama pognati te skripte. Vaše izkušnje so lahko drugačne z nekaterimi drugimi dostopnimi ukaznimi lupinami, kot so zsh, bash, itd.)

Očitno postane popravek za ‚#‘ nepotreben, če imate v jedru podporo za ‚#!‘. Pravzaprav je lahko nevaren, saj ustvarja zmedo pri ugotovitvi, kaj naj bi se zgodilo v primeru „#! /bin/sh“.

Uporaba ‚#!‘ bolj in bolj prevladuje. System V Release 4 pobira številne lastnosti z Berkeleya, vključno s to. Nekateri proizvajalci sistemov System V Release 3.2 nadgrajujejo na nekatere bolj vidne dobrote V.4 in vam poskušajo prepričati, da je to dovolj in, da ne potrebujete stvari, kot so pravi, delujoči tokovi ali dinamično nastavljivi parametri jedra.

XENIX ne podpira ‚#!‘. Ukazna lupina /bin/csh na Xenixu nima popravka za ‚#‘. Podpora za ‚#!‘ v Xenixu bi bila fina, a sam ne bi zadrževal diha med čakanjem nanjo.

5 Napredna vprašanja; radi jih vprašujejo ljudje, ki mislijo, da že poznajo vse odgovore

5.1 Kako berem znake s terminala, ne da bi bilo uporabniku treba pritisniti **RETURN**?

```
Subject: How do I read characters ... without requiring the user to hit RETURN?
Date: Thu Mar 18 17:16:55 EST 1993
```

Preverite način `cbreak` v BSD ali način `~ICANON` v SysV.

Če se sami ne želite spoprijeti s terminalskimi parametri (z uporabo sistemskega klica `ioctl(2)`), lahko prepustite delo programu `stty` - a to je počasno in neučinkovito, in včasih boste morali spremeniti kodo, da bo delala pravilno:

```
#include <stdio.h>
main()
{
    int c;

    printf("Pritisnite katerikoli znak za nadaljevanje\n");
    /*
     * funkcija ioctl() bi bila tukaj boljša;
     * le leni programerji delajo takole:
     */
    system("/bin/stty cbreak");        /* ali "stty raw" */
    c = getchar();
    system("/bin/stty -cbreak");
    printf("Hvala, ker ste vnesli %c.\n", c);

    exit(0);
}
```

Nekaj ljudi mi je poslalo različne bolj pravilne rešitve tega problema. Žal mi je, da jih ne morem vključiti, saj zares presegajo namen tega seznama.

Preveriti boste želeli tudi dokumentacijo prenosljive knjižnice zaslonskih funkcij, imenovane „`curses`“. Če vas zanima V/I enega samega znaka, vas pogosto zanima tudi neka vrsta kontrole prikaza na zaslonu, in knjižnica `curses` priskrbi različne prenosljive rutine za obe funkciji.

5.2 Kako preverim, če je treba prebrati znak, ne da bi ga zares prebral?

```
Subject: How do I check to see if there are characters to be read ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

V nekaterih različicah Unixa je mogoče preveriti, ali je v danem datotečnem deskriptorju kaj neprebranih znakov. V BSD-ju lahko uporabite `select(2)`. Uporabite lahko tudi `FIONREAD ioctl`, ki vrne število znakov, ki čakajo na prebranje, a to deluje le pri terminalih, cevovodih in vtičih. V System V Release 3 lahko uporabite `poll(2)`, a to deluje le na tokovih. V Xenixu - in torej na Unixu SysV r3.2 in poznejših - sistemski klic `rdchk()` poroča o tem ali se bo klic `read()` na danem datotečnem deskriptorju blokiral.

Ni načina, da bi preverili, ali so znaki dostopni za branje s kazalca `FILE`. (Lahko gledate po notranjih podatkovnih strukturah stdio, da bi videli, če je vmesni pomnilnik vhoda neprazen, a to ne bo delovalo, saj ne veste, kaj se bo zgodilo naslednjič, ko boste hoteli napolniti vmesni pomnilnik.)

Včasih ljudje vprašujejo to vprašanje z namenom, da bi napisali

```
if (znaki dostopni iz fd)
        read(fd, buf, sizeof buf);
```

in dobili učinek neblokovnega branja z `read`. To ni najboljši način za izvedbo tega, saj je možno, da bodo znaki dostopni, ko preverite dostopnost, a ne bodo več dostopni, ko pokličete `read`. Namesto tega prižgite zastavico `O_NDELAY` (ki se pod BSD imenuje tudi `FNDELAY`) z izbiro `F_SETFL` funkcije `fcntl(2)`. Starejši sistemi (Version 7, 4.1 BSD) nimajo `O_NDELAY`; na teh sistemih lahko najbližje neblokovnemu branju pridete z uporabo `alarm(2)`, da branju poteče čas.

5.3 Kako ugotovim ime odprte datoteke?

```
Subject: How do I find the name of an open file?
Date: Thu Mar 18 17:16:55 EST 1993
```

V splošnem je to pretežko. Datotečni deskriptor je lahko obešen na cevovod ali pty, in v tem primeru nima imena. Lahko je obešen na datoteko, ki je bila odstranjena. Lahko ima več imen, zaradi pravih ali simboličnih povezav.

Če morate to res storiti, in prepričajte se, da ste o tem razmislili na dolgo in široko in se odločili, da nimate druge izbire, lahko uporabite `find` z izbiro `-inum` in morda še `-xdev`, ali uporabite `ncheck`, ali še enkrat oponašajte funkcionalnost enega od teh orodij v vašem programu. Zavedajte se, da preiskovanje 600 megabytnega datotečnega sistema za datoteko, ki je morda sploh ni, traja nekaj časa.

5.4 Kako lahko tekoči program ugotovi svojo lastno pot?

```
Subject: How can an executing program determine its own pathname?
Date: Thu Mar 18 17:16:55 EST 1993
```

Vaš program lahko pogleda `argv[0]`; če se začenja z „/“, je to verjetno ime absolutne poti do vašega programa, sicer lahko vaš program pogleda v vsak imenik, imenovan v okoljski spremenljivki `PATH` in poskuša najti prvi imenik, ki vsebuje izvedljivo datoteko, katere ime se ujema s programovim `argv[0]` (ki je po dogovoru ime programa, ki se izvaja). Če združite ta imenik in vrednost `argv[0]`, imate verjetno pravo ime.

Ne morete pa biti prepričani, saj je povsem legalno, da en program izvede drugega z `exec()` s katerokoli vrednostjo `argv[0]`, ki si jo zaželi. Da `exec` izvaja nove programe z imenom izvedljive datoteke v `argv[0]`, je le dogovor.

Na primer, povsem hipotetični primer:

```
#include <stdio.h>
main()
{
    execl("/usr/games/rogue", "vi Disertacija", (char *)NULL);
}
```

Izvedeni program misli, da je njegovo ime (njegova vrednost `argv[0]`) „`vi Disertacija`“. (Tudi nekateri drugi programi lahko mislijo, da je ime programa, ki ga trenutno poganjate „`vi Disertacija`“, a seveda je to le hipotetični primer, zato tega ne poskušajte sami :-)

5.5 Kako uporabim popen() za odprtje procesa za branje in pisanje?

```
Subject: How do I use popen() to open a process for reading AND writing?
Date: Thu Mar 18 17:16:55 EST 1993
```

Problem, ko poskušate preusmeriti vhod in izhod poljubnemu suženjskemu procesu je, da lahko nastane mrtva zanka, če oba procesa hkrati čakata na še-ne-generiran vhod. Zanki se lahko izognemo tako, da **obe** strani upoštevata strog protokol brez mrtvih zank, toda, ker to zahteva sodelovanje med procesi, je neprimerno za knjižnično funkcijo, podobno `popen()`.

Distribucija „expect“ vključuje knjižnico funkcij, ki jih lahko C-jevski programer kliče neposredno. Ena izmed funkcij dela isto stvar kot popen za hkratno branje in pisanje. Uporablja ptys namesto cevi, in nima problemov z zaciklanjem. Prenosljiva je na BSD in SV. Glejte vprašanje 4.9 („Kako poženem passwd, ftp, telnet, tip in druge interaktivne programe v skriptu ukazne lupine v ozadju¿‘) za več podatkov o distribuciji expect.

5.6 Kako izvedem v C-ju `sleep()` za manj kot sekundo?

```
Subject: How do I sleep() in a C program for less than one second?
Date: Thu Mar 18 17:16:55 EST 1993
```

Najprej se zavedajte, da je vse, kar lahko določite **minimalna** količina zakasnitve; dejanska zakasnitev bo odvisna od upravniških zadev, kot je obremenitev sistema, in je lahko poljubno dolga, če nimate sreče.

Ne obstaja funkcija standardne knjižnice, na katero bi se lahko zanesli v vseh okolji za „dremanje“ (angl. napping, običajen izraz za kratke spance). Nekatera okolja priskrbijo funkcijo „usleep(*n*)“, ki zadrži izvajanje za *n* mikrosekund. Če vaše okolje ne podpira usleep(), je tukaj nekaj njenih implementacij za okolja BSD in System V.

Naslednja koda Douga Gwyna je prirejena z emulacijske podpore Systema V za 4BSD in izrablja sistemski klic select() na 4BSD-ju. Doug jo je prvotno imenoval „nap()“; vi jo boste verjetno želeli klicati „usleep()“:

```
/*
    usleep - podporna rutina za emulacijo sistemskih klicev 4.2BSD
    zadnja sprememba originalne verzije:  29. oktober 1984     D A Gwyn
*/

extern int        select();

int
usleep( usec )                          /* vrne 0, če je ok, sicer -1 */
    long               usec;            /* premor v mikrosekundah */
    {
    static struct                       /* 'timeval' */
            {
            long        tv_sec;         /* sekunde */
            long        tv_usec;        /* mikrosekunde */
            }   delay;          /* premor _select() */

    delay.tv_sec = usec / 1000000L;
    delay.tv_usec = usec % 1000000L;

    return select( 0, (long *)0, (long *)0, (long *)0, &delay );
    }
```

Na Unixih System V bi lahko to storili takole:

```
/*
podsekundni premori za System V - ali karkoli, kar ima poll()
Don Libes, 4/1/1991

BSD-jeva analogija te funkcije je definirana v mikrosekundah, medtem,
kot je poll() definiran v milisekundah.  Zaradi združljivosti, ta
```

```
funkcija priskrbi natančnost "po dolgem teku" tako, da oklesti prave
zahteve na milisekundo natančno in akumulira mikrosekunde med
posameznimi klici z idejo, da jo verjetno kličete v tesni zanki, in se
bo po dolgem teku napaka izničila.

Če je ne kličete v tesni zanki, potem skoraj gotovo ne potrebujete
mikrosekundne natančnosti in v tem primeru vam ni mar za mikrosekunde.
Če vam bi bilo mar, tako ali tako ne bi uporabljali Unixa, saj lahko
naključno prebavljanje sistema (npr. razporejanje) zmelje časomerilno
kodo.

Vrne 0 ob uspešnem premoru, -1 ob neuspešnem.

*/

#include <poll.h>

int
usleep(usec)
unsigned int usec;                /* mikrosekunde */
{
    static subtotal = 0;        /* mikrosekunde */
    int msec;                   /* milisekunde */

    /* 'foo' je tukaj le zato, ker so imele nekatere verzije 5.3
     * hrošča, pri katerem se prvi argument poll() preverja za
     * obstoj pravilnega pomnilniškega naslova, čeprav je drugi
     * argument enak 0.
     */
    struct pollfd foo;

    subtotal += usec;
    /* če je premor < 1 ms, ne naredi ničesar, a si zapomni */
    if (subtotal < 1000) return(0);
    msec = subtotal/1000;
    subtotal = subtotal%1000;
    return poll(&foo,(unsigned long)0,msec);
}
```

Še ena možnost za dremanje na System V in verjetno tudi drugih ne-BSD Unixih je paket `s5nap` Jona Zeeffa, objavljen v `comp.sources.misc`, volume 4. Ne potrebuje namestitve gonilnika naprave, a deluje brez napak, ko se namesti, (Njegova ločljivost je omejena z vrednostjo `HZ` v jedru, saj uporablja rutino `delay()` jedra.)

Mnogo novejših Unixov ima funkcijo `nanosleep`.

5.7 Kako pripravim skripte ‚setuid' ukazne lupine do delovanja?

```
Subject: How can I get setuid shell scripts to work?
Date: Thu Mar 18 17:16:55 EST 1993
```

[Ta odgovor je dolg, a to je zapleteno in pogosto zastavljeno vprašanje. Hvala Maartenu Litmaathu za ta odgovor in za spodaj omenjeni program „`indir`".]

1. Najprej predpostavimo, da ste na različici Unixa (npr. 4.3BSD ali SunOS), ki pozna tako imenovane „izvedljive skripte ukaznih lupin". Takšen skript se mora začeti s podobno vrstico:

   ```
   #!/bin/sh
   ```

   Skript se imenuje „izvedljivi", ker se tako kot resnična (binarna) izvedljiva datoteka začne s tako imenovano „magično številko", ki določa tip izvedljive datoteke. Glejte tudi razdelek 4.16 („Zakaj se nekateri skripti začnejo z „#!..."¿'). V našem primeru je ta številka enaka „`#!`" in OS vzame ostanek prve vrstice kot interpreter za skript, ki mu morda sledi 1 uvodna izbira kot je:

   ```
   #!/bin/sed -f
   ```

   Denimo, da se ta skript imenuje ‚`foo`' in se nahaja v imeniku `/bin`. Če potem napišete:

   ```
   foo arg1 arg2 arg3
   ```

   bo OS preuredil zadeve tako, kot bi napisali:

   ```
   /bin/sed -f /bin/foo arg1 arg2 arg3
   ```

   Vendar je tukaj neka razlika: če je prižgan bit setuid za ‚`foo`', bo spoštovan v prvi obliki ukaza; če zares vpišete drugo obliko, bo OS spoštoval bite z dovoljenji programa `/bin/sed`, ki seveda ni setuid.

2. Prav, toda kaj, če se moj skript ukazne lupine **ne** začne s takšno vrstico ‚`#!`', ali, če moj OS o tem nič ne ve? No, če ga ukazna lupina (ali kdorkoli drug) poskuša izvesti, bo OS vrnil indikacijo napake, saj se datoteka ne začne z veljavno magično številko. Ukazna lupina bo po sprejetju te indikacije **predpostavila**, da gre za skript ukazne lupine in mu dala še eno priložnost:

   ```
   /bin/sh lupinski_skript argumenti
   ```

   A zdaj smo že videli, da se v tem primeru bit setuid datoteke ‚`lupinski_skript`' **ne bo** spoštoval!

3. Prav, kaj pa varnostna tveganja pri lupinskih skriptih setuid? Dobro, denimo, da se skript imenuje ‚`/etc/skript_setuid`' in se začenja z vrstico:

   ```
   #!/bin/sh
   ```

   Zdaj pa poglejmo kaj se zgodi, če izvedemo naslednje ukaze:

   ```
   $ cd /tmp
   $ ln /etc/skript_setuid -i
   $ PATH=.
   $ -i
   ```

   Vemo, da bo zadnji ukaz preurejen v:

   ```
   /bin/sh -i
   ```

   Toda ta ukaz nam bo dal interaktivno ukazno lupino, ki bo z bitom setuid tekla, kot da bi jo pognal lastnik skripta!

   Na srečo lahko to varnostno luknjo zlahka zapremo, če napišemo v prvo vrstico:

   ```
   #!/bin/sh -
   ```

   Znak ‚`-`' označuje konec seznama izbir: naslednji argument ‚`-i`' bo vzet kot ime datoteke, iz katere naj se berejo ukazi, kot bi tudi moral biti!

4. Vendar pa obstajajo veliko resnejši problemi:

```
$ cd /tmp
$ ln /etc/skript_setuid temp
$ nice -20 temp &
$ mv moj_skript temp
```

Tretji ukaz bo preurejen v:

```
nice -20 /bin/sh - temp
```

Ker se ta ukaz izvaja počasi, bo morda lahko četrti ukaz zamenjal originalno datoteko ‚`temp`' s trojanskim konjem ‚`moj_skript`' **preden** ukazna lupina sploh odpre ‚`temp`'! Obstajajo 4 načini za krpanje te varnostne luknje:

(a) naj OS zažene skript setuid na drugačen, varen način - System V R4 in 4.4BSD uporabljata gonilnik `/dev/fd`, s katerim podata interpreterju datotečni deskriptor skripta;

(b) naj bo skript interpretiran posredno, skozi vmesnik, ki se prepriča, da je vse v redu, preden požene pravi interpreter - če uporabljate program ‚`indir`' iz arhiva `comp.sources.unix` bodo skripti setuid izgledali takole:

```
#!/bin/indir -u
#?/bin/sh /etc/skript_setuid
```

(c) napravite „binarni ovoj": pravo izvedljivo datoteko, ki je setuid in katere edina naloga je pognati interpreter z imenom skripta kot argumentom;

(d) napravite splošen „strežnik za skripte setuid", ki poskuša najti zahtevano ‚opravilo' v bazi podatkov veljavnih skriptov in ob uspehu požene pravi interpreter s pravimi argumenti.

5. Zdaj, ko smo se prepričali, da se interpretira pravilna datoteka, ali so prisotna še kakšna tveganja? Seveda! Za skripte ukazne lupine ne smete pozabiti eksplicitno nastavitev spremenljivke `PATH` na varno pot. Lahko ugotovite zakaj? Tukaj je še spremenljivka `IFS`, ki lahko povzroča težave, če ni pravilno nastavljena. Tudi druge okoljske spremenljivke lahko ogrozijo varnost sistema, npr. `SHELL` ... Nadalje se morate prepričati, da ukazi v skriptih ne dovoljujejo interaktivnih ubežnih zaporedij ukazne lupine! Potem je tukaj umask, ki je lahko nastavljen na kaj čudnega ...

   Et cetera. Zavedati se morate, da skript setuid „podeduje" vse hrošče in varnostna tveganja ukazov, ki jih kliče!

Po vsem tem dobimo vtis, da so lupinski skripti setuid precej tvegan posel! Morda bo za vas bolje programirati v C-ju!

5.8 Kako lahko ugotovim, kateri uporabnik ali proces ima odprto datoteko ali uporablja določen datotečni sistem (da ga lahko odklopim)?

```
Subject: How can I find out which user or process has a file open ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Uporabite `fuser` (system V), `fstat` (BSD), `ofiles` (v javni lasti) ali `pff` (v javni lasti). Ti programi vam bodo povedali različne stvari o procesih, ki uporabljajo določene datoteke.

V arhivih `comp.sources.unix`, volume 18, najdete prenos `fstat` z 4.3 BSD na Dynix, SunOS in Ultrix.

`pff` je del paketa `kstuff` in deluje na kar nekaj sistemih. Navodila za nabavo `kstuff` najdete v vprašanju 4.10 („Kako lahko v skriptu ali programu ugotovim ID procesa programa z določenim imenom¿').

Obveščen sem bil, da obstaja tudi program, imenovan `lsof`, a ne vem, kje se ga dobi.

Michael Fink <`Michael.Fink@uibk.ac.at`> dodaja:

Če ne morete odmestiti datotečnega sistema (z `umount`), za katerega zgornja orodja ne poročajo o odprtih datotekah, se prepričajte, da datotečni sistem, ki ga poskušate odmeščati ne vsebuje aktivnih točk nameščanja (uporabite `df(1)`).

5.9 Kako izsledim ljudi, ki me tipajo (s **finger**)?

```
Subject: How do I keep track of people who are fingering me?
From: Jonathan I. Kamens
From: malenovi@plains.NoDak.edu (Nikola Malenović)
Date: Thu, 29 Sep 1994 07:28:37 -0400
```

V splošnem ne morete ugotoviti userid nekoga, ki vas tipa z oddaljenega stroja. Morda lahko ugotovite stroj, s katerega prihajajo oddaljene zahteve. Ena od možnosti, če jo vaš sistem podpira, in, če je tipalnemu strežniku (angl. finger daemon) prav, jo, da naredite vašo datoteko `.plan` za „imenovano cev“ (angl. named pipe) namesto običajno datoteko. (Za to uporabite ‚`mknod`‘.)

Potem lahko poženete program, ki bo odprl vašo datoteko `.plan` za pisanje; odpiranje bo blokirano dokler nek drugi proces (namreč `fingerd`) ne odpre `.plan` za branje. Zdaj lahko skozi to pipo pošljete, kar pač želite, kar vam omogoča, da prikažete različno informacijo `.plan` vsakič, ko vas nekdo potipa. Eden od programov za to je paket „`planner`“, objavljen v volume 41 arhivov `comp.sources.misc`.

Seveda to sploh ne bo delovalo, če vaš sistem ne podpira imenovanih cevi ali, če vaš lokalni `fingerd` vztraja, da imate navadne datoteke `.plan`.

Vaš program lahko tudi izkoristi priložnost in pogleda v izhod programa „`netstat`“, ter s tem ugotovi, odkod prihaja zahteva za `finger`, toda to vam ne bo izdalo identitete oddaljenega uporabnika.

Če želite dobiti oddaljeni userid, mora oddaljena stran poganjati identifikacijski strežnik, kot je RFC 931. Trenutno obstajajo tri implementacije RFC 931 za popularne stroje BSD, in nekaj aplikacij (kot `ftpd` z wuarchive), ki podpirajo ta strežnik. Za več informacij se priključite elektronskemu spisku `rfc931-users` z običajno zahtevo „`subscribe`“ na `rfc931-users-request@kramden.acf.nyu.edu`.

Glede tega odgovora obstajajo tri opozorila. Prvo je, da mnogi sistemi NFS napačno prepoznajo imenovano cev. To pomeni, da bo poskus branja cevi na drugem stroju lahko blokiral, dokler ne poteče predviden čas, ali videl cev kot datoteko dolžine 0, in je ne bo nikoli izpisal.

Drugi problem je, da na veliko sistemih strežnik `fingerd` preveri, ali datoteka `.plan` vsebuje podatke (in je bralna), preden jo poskuša brati. V tem primeru bodo oddaljeni klici `finger` popolnoma zgrešili vašo datoteko `.plan`.

Tretji problem je, da imajo sistemi, ki podpirajo imenovane cevi, na voljo v danem času le fiksno (končno) število leteh - preverite nastavitveno datoteko jedra in izbiro `FIFOCNT`. Če število cevi na sistemu prekorači vrednost `FIFOCNT`, sistem prepreči izdelavo novih cevi, dokler nekdo ne sprosti virov. Razlog za to je, da je vmesni pomnilnik odmerjen v pomnilniku, ki se ne preklaplja (angl. buffers are allocated in a non-paged memory).

5.10 Je mogoče ponovno priključiti proces na terminal, ko je bil ta odklopljen, tj. po zagonu programa v ozadju in odjavi?

```
Subject: Is it possible to reconnect a process to a terminal ... ?
Date: Thu Mar 18 17:16:55 EST 1993
```

Večina različic Unixa ne podpira „odklopa“ in „priklopa“ procesov (angl. detaching/attaching processes), kot ju podpirajo operacijski sistemi kot sta VMS in Multics. Vendar obstajajo prosto dostopni paketi, s katerimi lahko poženete procese na takšen način, da se lahko pozneje spet pritrdijo na terminal.

- Prvi tak paket je „`screen`“, ki je opisan v arhivih `comp.sources.unix` kot „Screen, multiple windows on a CRT“ (slov. „Zaslon, več oken na CRT“ - glejte paket `screen-3.2` v `comp.sources.misc`, volume 28). Ta paket bo tekel vsaj na sistemih BSD, System V r3.2 in SCO UNIX.

- Drugi je „`pty`", ki je v arhivih `comp.sources.unix` opisan kot „Run a program under a pty session" (slov. „Poženite program v seji pty" - glejte `pty` v volume 23). Načrtovan je le za uporabo v sistemih, podobnih BSD.
- Tretji je „`dislocate`", ki je skript, priložen distribuciji `expect`. Za razliko od prejšnjih dveh bi moral ta teči na vseh različicah Unixa. Podrobnosti o tem, kako dobite `expect` najdete v vprašanju 4.9 („Kako poženem passwd, ftp, telnet, tip in druge interaktivne programe v skriptu ukazne lupine v ozadju¿").

Nobeden od teh paketov ne deluje retroaktivno, se pravi, da morate pognati proces pod pripomočkom `screen` ali `pty`, če ga želite odklopiti in priklopiti.

5.11 Je mogoče „vohuniti" na terminalu, gledajoč izhod, ki se prikazuje na drugem terminalu?

```
Subject: Is it possible to "spy" on a terminal ... ?
Date: Wed, 28 Dec 1994 18:35:00 -0500
```

Za to obstaja več različnih poti, čeprav nobena od njih ni popolna:

- Pripomoček `kibitz` dovoljuje dvema (ali več) človekoma, da sta v stiku prek ukazne lupine (ali poljubnega programa). Uporaba vključuje:
  - opazovanje ali pomoč pri terminalski seji druge osebe;
  - beleženje pogovora z možnostjo pomika nazaj, shranjevanje pogovora, in celo spreminjanje med tekom;
  - moštveno igranje iger, urejanje dokumentov ali druga opravila, pri katerih ima vsaka oseba določene kvalitete in določene pomanjkljivosti, ki se komplementirajo.

  `kibitz` pride kot del distribucije `expect`. Glejte 4.9 („Kako poženem passwd, ftp, telnet, tip in druge interaktivne programe v skriptu ukazne lupine v ozadju¿").

  Za uporabo pripomočka `kibitz` potrebujete dovoljenje osebe, ki jo želite vohuniti. Vohunjenje brez privolitve zahteva manj prijetne pristope:
- Napišete lahko program, ki brska po strukturah jedra in opazuje izhodni vmesni pomnilnik spornega terminala, ter pri tem izpisuje znake, kot se izpisujejo na izhod. To očitno ni nekaj, kar naj bi poskušal kdorkoli, ki nima izkušenj pri delu okoli jedra operacijskega sistema Unix. Še več, katerokoli metodo boste že uporabili, bo verjetno zelo neprenosljiva.
- Če želite to početi ves čas na določenem pritrjenem terminalu (npr. če želite, da bodo lahko operaterji preverili zaslonski terminal stroja z drugih strojev), lahko dejansko povežete monitor s terminalskim kablom. Na primer, vtaknite izhod monitorja v serijska vrata drugega stroja, in na teh vratih poženite program, ki bo nekam shranil svoj vhod in ga potem prenesel na **druga** vrata, tista, ki gredo zares na fizični terminal. Če to počnete, se morate prepričati, da se ves izhod terminala pošilja nazaj po žici, če povezujete le žice računalnik->terminal, to ni problem. To ni nekaj, kar bi lahko počel nekdo, ki ni domač z napeljavo terminalskih žic in podobno.
- Zadnja različica paketa `screen` vključuje večuporabniški način. Nekaj podatkov o paketu `screen` najdete v vprašanju 5.10 („Je mogoče ponovno priključiti proces na terminal, ko je bil ta odklopljen, tj. po zagonu programa v ozadju in odjavi¿").
- Če ima sistem, ki ga uporabljate, tokove (SunOS, SVR4), vam kot nasvet priporočamo program, objavljen v volume 28 arhiva skupine `comp.sources.misc`. **In** ni ga treba najprej zagnati (vnaprej morate nastaviti vaš sistem, da samodejno potisne modul `advise` na tok vsakič, ko se odpre tty ali pty).

6 Vprašanja, povezana z različnimi ukaznimi lupinami in njihovimi razlikami

6.1 Se lahko ukazne lupine klasificirajo v kategorije?

```
Subject: Can shells be classified into categories?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
Date: Wed, 7 Oct 92 14:28:18 -0500
```

V splošnem obstajata dva glavna razreda ukaznih lupin. Prvi razred sestavljajo lupine, izpeljane iz Bournove ukazne lupine, in vključuje `sh`, `ksh`, `bash` in `zsh`. Drugi razred sestavljajo ukazne lupine, izpeljane iz C-jevske in vključuje `csh` in `tcsh`. Kot dodatek je tukaj še `rc`, za katero večina ljudi meni, da je „razred zase“, čeprav nekateri lahko ugovarjajo, da `rc` spada v razred Bournovih ukaznih lupin.

Z zgornjo klasifikacijo in nekaj previdnosti je mogoče napisati skripte, ki bodo delovali v vseh ukaznih lupinah Bournove lupinske kategorije, in napisati druge skripte, ki bodo delovali v vseh ukaznih lupinah C-jevske lupinske kategorije.

6.2 Kako „vključim“ en skript ukazne lupine iz drugega lupinskega skripta?

```
Subject: How do I "include" one shell script from within another shell script?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
Date: Wed, 7 Oct 92 14:28:18 -0500
```

Vse ukazne lupine Bournove kategorije (vključno z `rc`) uporabljajo ukaz „`.`“. Vse ukazne lupine C-jevske kategorije uporabljajo „`source`“.

6.3 Ali vse ukazne lupine podpirajo vzdevke (angl. aliases)? Lahko uporabimo tudi kaj drugega?

```
Subject: Do all shells have aliases?  Is there something else that can be used?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
Date: Wed, 7 Oct 92 14:28:18 -0500
```

Vse pomembnejše ukazne lupine, razen `sh`, podpirajo vzdevke, a vse ne delajo z njimi na enak način. Na primer, nekateri ne sprejemajo argumentov.

Čeprav niso strogo ekvivalentne, imajo funkcije ukaznih lupin (ki obstajajo v večini ukaznih lupin Bournove kategorije) skoraj isto funkcionalnost, kot vzdevki. Funkcije v ukaznih lupinah lahko počnejo stvari, ki jih vzdevki ne morejo. Funkcije ne obstajajo v Bournovih ukaznih lupinah, izpeljanih iz Unixa Version 7, ki vključuje System III in BSD 4.2. BSD 4.3 in System V imata ukazne lupine s podporo funkcij.

Uporabite `unalias` za odstranitev vzdevkov (nastalih z `alias`) in `unset` za odstranitev funkcij.

6.4 Kako se prirejajo spremenljivke ukaznih lupin?

```
Subject: How are shell variables assigned?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
Date: Wed, 7 Oct 92 14:28:18 -0500
```

Ukazne lupine C-jevske kategorije uporabljajo

```
set spremenljivka=vrednost
```

za spremenljivke, lokalne ukazni lupini, in

```
setenv spremenljivka vrednost
```

za okoljske spremenljivke. Spremenljivk se v teh ukaznih lupinah znebite z uporabo ukazov `unset` in `unsetenv`.

Ukazne lupine Bournove kategorije uporabljajo

```
spremenljivka=vrednost
```

in morda potrebujejo tudi

```
export IME_SPREMENLJIVKE
```

da postavijo spremenljivko v okolje. Spremenljivk se znebite z `unset`.

6.5 Kako ugotovim, ali poganjam interaktivno ukazno lupino?

```
Subject: How can I tell if I am running an interactive shell?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
From: dws@ssec.wisc.edu (DaviD W. Sanderson)
Date: Fri, 23 Oct 92 11:59:19 -0600
```

V ukaznih lupinah C-jevske kategorije, poglejte spremenljivko `$prompt`.

V Bournovi kategoriji ukaznih lupin lahko pogledate spremenljivko `$PS1`, vendar je bolj pogledati spremenljivko `$-`. Če `$-` vsebuje ‚`-i`', je ukazna lupina interaktivna. Preverite takole:

```
case $- in
*i*)    # izvedi stvari za interaktivno ukazno lupino
        ;;
*)      # izvedi stvari za neinteraktivno ukazno lupino
        ;;
esac
```

6.6 Katere datoteke „s piko" uporabljajo različne ukazne lupine?

```
Subject: What "dot" files do the various shells use?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
From: tmb@idiap.ch (Thomas M. Breuel)
Date: Wed, 28 Oct 92 03:30:36 +0100
```

Čeprav ta seznam morda ni popoln, ponuja večino informacij.

csh

Nekatere različice imajo sistemski datoteki `.cshrc` in `.login`. Vsaka različica jih postavlja na različna mesta.

Ob nastopu (v tem vrstnem redu):

- `.cshrc` - vedno; razen ob uporabi izbire `-f`.
- `.login` - prijavne ukazne lupine.

Ob prekinitvi:

- `.logout` - prijavne ukazne lupine.

Ostalo:

- `.history` - shranjuje zgodovino (temelji na `$savehist`).

tcsh

Ob nastopu (v tem vrstnem redu):

- `/etc/csh.cshrc` - vedno.
- `/etc/csh.login` - prijavne ukazne lupine.
- `.tcshrc` - vedno.
- `.cshrc` - če manjka `.tcshrc`.
- `.login` - prijavne ukazne lupine.

Ob prekinitvi:

- `.logout` - prijavne ukazne lupine.

Ostalo:

- `.history` - shranjuje zgodovino (temelji na `$savehist`).
- `.cshdirs` - shranjuje sklad imenikov.

sh

Ob nastopu (v tem vrstnem redu):

- `/etc/profile` - prijavne ukazne lupine.
- `.profile` - prijavne ukazne lupine.

Ob prekinitvi:

- katerikoli ukaz (ali skript), določen z ukazom:

```
trap "ukaz" 0
```

ksh

Ob nastopu (v tem vrstnem redu):

- `/etc/profile` - prijavne ukazne lupine.
- `.profile` - prijavne ukazne lupine; razen ob uporabi izbire `-p`.
- `$ENV` - vedno, če je nastavljena; razen ob uporabi izbire `-p`.
- `/etc/suid_profile` - ob uporabi izbire `-p`.

Ob prekinitvi:

- katerikoli ukaz (ali skript), določen z ukazom:

```
trap "ukaz" 0
```

bash

Ob nastopu (v tem vrstnem redu):

- `/etc/profile` - prijavne ukazne lupine.
- `.bash_profile` - prijavne ukazne lupine.
- `.profile` - prijavne ukazne lupine, če ni datoteke `.bash_profile`.
- `.bashrc` - interaktivne neprijavne ukazne lupine.
- `$ENV` - vedno, če je nastavljena.

Ob prekinitvi:

- `.bash_logout` - prijavne ukazne lupine.

Drugo:

- `.inputrc` - inicializacija branja vrstic.

zsh

Ob nastopu (v tem vrstnem redu):

- `.zshenv` - vedno, razen ob določitvi izbire `-f`.
- `.zprofile` - prijavne ukazne lupine.
- `.zshrc` - interaktivne ukazne lupine, razen ob določitvi izbire `-f`.
- `.zlogin` - prijavne ukazne lupine.

Ob prekinitvi:

- `.zlogout` - prijavne ukazne lupine.

rc

Ob nastopu:

- `.rcrc` - prijavne ukazne lupine.

6.7 Zanima me več o razlikah med različnimi ukaznimi lupinami.

```
Subject: I would like to know more about the differences ... ?
From: wicks@dcdmjw.fnal.gov (Matthew Wicks)
Date: Wed, 7 Oct 92 14:28:18 -0500
```

Zelo podrobna primerjava ukaznih lupin `sh`, `csh`, `tcsh`, `ksh`, `bash`, `zsh`, in `rc` je v datoteki dostopna po anonimnem FTP-ju z več naslovov:

- <ftp://ftp.uwp.edu/pub/vi/docs/shell-100.BetaA.Z>
- <ftp://utsun.s.u-tokyo.ac.jp/misc/vi-archive/docs/shell-100.BetaA.Z>

Ta datoteka primerja zastavice, programsko skladnjo, vhodno/izhodne preusmeritve in parametre/okoljske spremenljivke ukaznih lupin. Ne ukvarja pa se s tem, katere datoteke „s piko“ se uporabljajo in dedovanjem okoljskih spremenljivk in funkcij.

7 Pregled različic Unixa

7.1 Opozorilo in uvod

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Tue Aug 15 15:14:00 EDT 1995
X-Version: 2.9
```

Opozorilo bralcem: Rad bi osvežil ta PZV s spletnimi kazalci na različne Unixe, ki jih omenjam. Ne oklevajte s pošiljanjem, slej ko prej bom končal osvežitev tega dela. E-pošta: `lew@bnr.ca`.

Nadaljnje besedilo je ponujeno brez zagotovitve točnosti ali popolnosti. Storil sem, kar sem lahko storil v času, ki ga imam na voljo, pogosto z nasprotujočimi si informacijami, in delo še vedno napreduje. Upam, da bom nadaljeval z izboljševanjem tega povzetka. Komentarji in popravki so dobrodošli: `lew@bnr.ca`.

Najprej kratka definicija. Z imenom „Unix“ mislimo operacijski sistem, tipično napisan v C-ju, s hierarhično urejenim datotečnim sistemom, integracijo V/I za datoteke in naprave, katerega vmesnik sistemskih klicev vključuje storitve kot sta `fork()`, `pipe()`, in katerega uporabniški vmesnik vključuje orodja kot so `cc`, `troff`, `grep`, `awk`, in izbiro ukazne školjke. Opazite, da je „UNIX“ registrirana blagovna znamka podjetja USL (AT&T), zdaj v lasti konzorcija X/Open, a bo tukaj uporabljana v svojem generičnem pomenu.

Večina Unixov je izpeljanih bolj ali manj neposredno iz izvorne kode, napisane v AT&T (zdaj Novell, verjetno je v večini še vedno kakšen ostanek prvotne kode prve različice v C-ju), a obstajajo tudi kloni (se pravi, z Unixom združljivi sistemi brez kode AT&T).

Kot dodatek, obstajajo okolja, podobna Unixu (npr. VOS), ki počivajo na drugih operacijskih sistemih (OS), in OS, ki jih je navdihnil Unix (da, tudi DOS!). Teh tukaj ne obravnavamo. Le malo pišemo o Unixih, ki delujejo v realnem času (angl. real-time Unices), čeprav načrtujemo več.

Unix pride v izredno različnih okusih. To je v veliki meri posledica dostopnosti izvorne kode in preprostosti prenosa in spremembe Unixa. Tipično, izdelovalec Unixa začne z eno osnovnih vrst (glejte spodaj), vzame ideje/kodo od druge večje vrste, doda in spremeni veliko stvari, itd. Rezultat je še ena nova vrsta Unixa. Dandanes je na voljo dobesedno na stotine Unixov, najbližja stvar standardnemu Unixu je (po definiciji) System V.

Ta odgovor je bil napisan predvsem iz podatkov na Mreži in e-pošti. Nekaj posebnih virov je omenjenih v ustreznih razdelkih.

Priznanja (poleg virov): `pat@bnr.ca`, `guy@auspex.com`, `pen@lysator.liu.se`, `mikes@ingres.com`, `mjd@saul.cis.upenn.edu`, `root%candle.uucp@ls.com`, `ee@atbull.bull.co.at`, `Aaron_Dailey@stortek.com`, `ralph@dci.pinetree.org`, `sbdah@mcshh.hanse.de`, `macmach@andrew.cmu.edu`, `jwa@alw.nih.gov` [4.4BSD], `roeber@axpvms.cern.ch`, `bob@pta.pyramid.com.au`, `bad@flatlin.ka.sub.org`, `m5@vail.tivoli.com`, `dan@fch.wimsey.bc.ca`, `jlbrand@uswnvg.com`, `jpazer@usl.com`, `ym@satelnet.org`, `merritt@gendev.slc.paramax.com`, `quinlan@yggdrasil.com`, `steve@rudolph.ssd.csd.harris.com`, `bud@heinous.isca.uiowa.edu`, `pcu@umich.edu`, `Dan_Menchaca@quickmail.apple.com`, `D.Lamptey@sheffield.ac.uk`, `derekn@vw.ece.cmu.edu`, `gordon@PowerOpen.org`, `romain@pyramid.com`, `rzm@dain.oso.chalmers.se`, `chen@adi.com`, `tbm@tci002.uibk.ac.at`, `sllewis@nando.net`, `edwin@modcomp.demon.co.uk`, veliko ljudem, ki sem jih še pozabil našteti, in vsem drugim ljudem, katerih sporočila berem. Najlepša hvala!

7.2 Zelo kratek pogled na zgodovino Unixa

Zgodovina Unixa se začne davnega leta 1969 na slavnem „malo-uporabljanem računalniku PDP-7 v kotu“ na katerem so Ken Thompson, Dennis Ritchie (črka ‚R‘ v „K&R“) in drugi začeli delo na tem, kar je potem postalo Unix. Ime

„Unix“ je bilo mišljeno kot besedna igra imena obstoječega operacijskega sistema „Multics“ (in se je najprej pisalo „Unics“ - iz angl. UNiplexed Information and Computing System).

Prvih 10 let je bil razvoj Unixa pravzaprav omejen na Bell Labs. Te prve različice so bile označene kot „Version n“ ali „Nth Edition“ (priročnikov), in so delovale na računalnikih DEC PDP-11 (16 bitov) in poznejših VAX-ih (32 bitov). Pomembnejše različice vključujejo:

- V1 (1971): Prva različica Unixa, v zbirniku na PDP-11/20. Vključevala je datotečni sistem, `fork()`, `roff`, `ed`. Uporabljana je bila kot urejevalnik besedila za pripravo patentov. Klic `pipe()` se je pojavil prvič v V2!
- V4 (1973): Prepisan v C, kar je verjetno najpomembnejši dogodek v zgodovini tega OS: pomeni, da se Unix lahko prenaša na novo strojno opremo v nekaj mesecih, spremembe pa so preproste. Programski jezik C je bil prvotno načrtovan za operacijski sistem Unix, in torej obstaja močna sinergija med C-jem in Unixom.
- V6 (1975): Prva različica Unixa, široko dostopna zunaj ustanove Bell Labs (posebej na univerzah). To je bil tudi začetek raznolikosti in popularnosti Unixa. 1.xBSD (PDP-11) je bil izpeljan iz te različice. J. Lions je objavil svoj „Komentar o operacijskem sistemu Unix“, ki temelji na V6.
- V7 (1979): Za veliko ljudi je to „zadnji pravi Unix“, ali „izboljšanje glede na vse prejšnje in prihodnje Unixe“ [Bourne]. Vključeval je popoln K&R C, `uucp`, Bournovo ukazno lupino. V7 je bil prenesen na VAX kot 32V. Jedro V7 je bilo veliko borih 40 Kbytov! Tukaj so (za referenco) sistemski klici V7: `_exit`, `access`, `acct`, `alarm`, `brk`, `chdir`, `chmod`, `chown`, `chroot`, `close`, `creat`, `dup`, `dup2`, `exec*`, `exit`, `fork`, `fstat`, `ftime`, `getegid`, `geteuid`, `getgid`, `getpid`, `getuid`, `gtty`, `indir`, `ioctl`, `kill`, `link`, `lock`, `lseek`, `mknod`, `mount`, `mpxcall`, `nice`, `open`, `pause`, `phys`, `pipe`, `pkoff`, `pkon`, `profil`, `ptrace`, `read`, `sbrk`, `setgid`, `setuid`, `signal`, `stat`, `stime`, `stty`, `sync`, `tell`, `time`, `times`, `umask`, `umount`, `unlink`, `utime`, `wait`, `write`.

Omenjene različice Vn je razvila skupina *Computer Research Group* (CRG) ustanove Bell Labs. Druga skupina, *Unix System Group* (USG), je odgovorna za podporo. Tretja skupina pri Bell Labs je bila tudi vpletena v razvijanje Unixa, *Programmer's WorkBench* (PWB), kateri se moramo zahvaliti, na primer, za `sccs`, poimenske cevi (angl. named pipes) in druge pomembne zamisli. Obe skupini sta se leta 1983 združili v *Unix System Development Lab*.

Druga različica Unixa je bila CB Unix (Columbus Unix) s kolumbijske veje Bell Labs, odgovorne za sisteme za podporo operacij. Njihov glavni prispevek je del SV IPC.

Delo na Unixu v Bell Labs se je v osemdesetih nadaljevalo. Serije V je naprej razvijala skupina CRG (Stroustrup omenja V10 v drugi izdaji njegove knjige o C++), a sicer ne slišimo veliko o tem. Podjetje, zdaj odgovorno za Unix (System V) se imenuje Unix System Laboratories (USL) in je bilo v večinski lasti koncerna AT&T do začetka leta 1993, ko ga je kupil Novell. Konec leta 1993 je Novell odstopil pravice do uporabe blagovne znamke „UNIX“ mednarodnemu združenju X/Open.

A z Unixom se je veliko dogajalo zunaj AT&T, posebej na univerzi Berkeley (s koder prihaja druga večja vrsta Unixa). Proizvajalci (predvsem delovnih postaj) so prav tako prispevali veliko (npr. NFS podjetja Sun).

Knjiga „Life with Unix“ Dona Libesa in Sandy Ressler je fascinantno branje za vsakogar, ki ga zanima Unix, in pokriva veliko zgodovine, posredovanj, itd. Dobršen del tega razdelka je povzetega po tej knjigi.

7.3 Glavne vrste Unixa

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Mon Jan  9 16:59:14 EST 1995
X-Version: 2.7
```

Naslednje je bolj ali manj pogled zgodnjih 90-ih.

Do nedavnega sta obstajali dve glavni vrsti Unixa: System V (pet) podjetja AT&T, in Berkeley Software Distribution (BSD). SVR4 je pravzaprav mešanica obeh teh vrst. Konec 1991 je bil pri Open Software Foundation izdan OSF/1 (kot neposreden tekmec Systemu V) in lahko (prihodnost bo povedala) spremeni to sliko.

V nadaljevanju sledi pregled glavnih izdaj in lastnosti sistemov System V, BSD in OSF/1.

System V podjetja AT&T. Tipično na strojni opremi Intela. Najbolj prenašani Unix, tipično z razširitvami BSD (`csh`, nadzor opravil, `termcap`, `curses`, `vi`, simbolične povezave). Evolucijo Systema V zdaj nadzoruje Unix International (UI). Članstvo UI vključuje AT&T, Sun, ... Novičarski skupini: `comp.unix.sysv286` in `comp.unix.sysv386`. Glavne izdaje:

- System III (1982): prvi komercialni Unix podjetja AT&T
  - vrste FIFO (imenovane cevi) (pozneje?)
- System V (1983):
  - paket IPC (`shm`, `msg`, `sem`)
- SVR2 (1984):
  - funkcije ukazne lupine (`sh`)
  - SVID (definicija vmesnika System V, angl. System V Interface Definition)
- SVR3 (1986) za platforme ?:
  - tokovi (STREAMS, navdih pri V8), `poll()`, TLI (omrežno programje)
  - RFS
  - deljene programske knjižnice
  - SVID 2
  - prenos pomnilniških strani na zahtevo (angl. demand paging), če strojna oprema to podpira
- SVR3.2:
  - združitev s Xenixom (Intel 80386)
  - omreženost
- SVR4 (1988), glavni tok implementacij Unixa, mešanica sistemov System V, BSD, in SunOS.
  - S SVR3: sistemska administracija, terminal I/F, tiskalnik (z BSD-ja?), RFS, STREAMS, uucp
  - Z BSD-ja: FFS, TCP/IP, vtiči (angl. sockets), `select()`, `csh`
  - S SunOS: NFS, grafični vmesnik OpenLook, X11/NeWS, podsistem navideznega pomnilnika z datotekami, ki ustrezajo pomnilniku, deljene knjižnice (!= od SVR3?)
  - `ksh`
  - ANSI C
  - internacionalizacija (8-bitno čist)
  - ABI (Application Binary Interface - rutine namesto trapov (? angl. routines instead of traps))
  - POSIX, X/Open, SVID3
- SVR4.1
  - asinhron V/I (s SunOS?)
- SVR4.2 (temelji na SVR4.1ES)

  - datotečni sistem Veritas, ACL-ji
  - dinamično nalaganje modulov jedra
- Prihodnost:
  - SVR4 MP (večprocesorski)
  - uporaba mikrojedra Chorus

Berkeley Software Distribution (BSD, distribucija programja z Berkeleya). Tipična na VAX-ih, RISC-ih, več delovnih postajah. Zdaj bolj dinamične, raziskovalne različice kot System V. BSD Unix je odgovoren za večino priljubljenosti Unixa. Večina izboljšav Unixa se je začela tukaj. Skupina, odgovorna na UCB (University of California, Berkeley) je *Computer System Research Group* (CSRG). Ukinili so jo leta 1992. Novičarska skupina: `comp.unix.bsd`. Glavne izdaje:

- 2.xBSD (1978) za PDP-11, še vedno pomemben? (2.11BSD je bil izdan 1992!).
  - `csh`
- 3BSD (1978):
  - navidezni pomnilnik
- 4.?BSD:
  - `termcap`, `curses`
  - `vi`
- 4.0BSD (1980).
- 4.1BSD (?): osnova za poznejše različice AT&T CRG
  - nadzor opravil
  - samodejna nastavitev jedra
  - `vfork()`
- 4.2BSD (1983):
  - TCP/IP, vtiči, ethernet
  - datotečni sistem UFS: dolga imena datotek, simbolične povezave
  - novi zanesljivi signali (v SVR3 zdaj zanesljivi signali 4.1)
  - `select()`
- 4.3BSD (1986) za VAX, ?:
  - TCP/IP, vtiči, ethernet
  - datotečni sistem UFS: dolga imena datotek, simbolične povezave
  - novi zanesljivi signali (v SVR3 zdaj zanesljivi signali 4.1)
  - `select()`
- 4.3 Tahoe (1988): 4.3BSD z izvorno kodo, podpora za Tahoe (32-bitni supermini)
  - Fat FFS
  - novi algoritmi za TCP
- 4.3 Reno (1990) za VAX, Tahoe, HP 9000/300:

  - večina standarda POSIX 1003.1
  - novi algoritmi za TCP
  - omrežni datotečni sistem NFS (od Suna)
  - pomnilniški datotečni sistem MFS
  - OSI: TP4, CLNP, ISODE-ov FTAM, VT in X.500; SLIP
  - Kerberos

- Trakova Net1 (?) in Net2 (junij 1991): del BSD, za katerega ni potrebna pravna zaščita razširjanja USL
- 4.4BSD (alpha, junij 1992) za HP 9000/300, Sparc, 386, DEC, drugi; niti VAX, niti Tahoe; dve različici, lahka (vsebina Net2 plus, popravki in nove arhitekture) in oteženo (vse, potrebuje licenco USL):
  - nov navidezni pomnilniški sistem (VMS), ki temelji na Mach 2.5
  - vmesnik navideznega datotečnega sistema, logično strukturiran datotečni sistem, velikost lokalnega datotečnega sistema do 2ˆ63, NFS (prosto razširljiv, deluje s Suni, čez UDP ali TCP)
  - podpora omrežjem ISO/OSI (izpeljanim iz ISODE): TP4/CLNP/802.3 in TP0/CONS/X.25, seje in zgornje v uporabniškem prostoru (angl. session and above in user space); FTAM, VT, X.500.
  - večina POSIX.1 (posebej nov terminalski gonilnik a la SV), veliko od POSIX.2, izboljšan nadzor opravil; glave ANSI C
  - Kerberos integriran z večino sistema (vključno z NFS)
  - izboljšave TCP/IP (vključno s predvidevanjem glave (angl. header prediction), SLIP)
  - pomembne spremembe jedra (nov dogovor o sistemskih klicih, ...)
  - druge izboljšave: vrste FIFO, zaklepanje datoteke v določenem območju bytov (angl. byte-range file locking)

Ustanova *Open Software Foundation* (OSF) je konec leta 1991 izdala svoj Unix imenovan OSF/1. Še vedno potrebuje licenco SVR2. Združljiv/podrejen SVID 2 (in prihajajoči 3), POSIX, X/Open, itd. Člani OSF vključujejo Apollo, Dec, HP, IBM, ...

- OSF/1 (1991):
  - počiva na jedru Mach 2.5
  - simetrično multiprocesiranje, paralelizirano jedro, niti
  - logični diski, zrcaljenje diska (angl. disk mirroring), UFS (domoroden), S5 FS, NFS
  - izboljšana varnost (B1 z nekaterimi lastnostmi B2, B3; ali C2), 4.3BSD admin
  - STREAMS, TLI/XTI, vtiči (angl. sockets)
  - deljene knjižnice, dinamični nalagalnik (vključno z jedrom)
  - grafični vmesnik Motif
- izdaja 1.3 (junij 1994)
  - OSF/1 MK (mikro-jedro) temelji na Mach 3.0
  - zadošča trenutnemu osnutku specifikacije Specification 1170 (mišljeno za standardizacijo v procesu Fast Track konzorcija X/Open)
  - Data Capture I/F, Common Data Link I/F,
  - podpora ISO 10646 (Unicode) in 64-bitna podpora

Ta seznam večjih vrst bi moral verjetno vključevati tudi Xenix (Microsoft), ki je bil osnova za mnoge prenose. Izpeljan je bil iz V7, S III in končno System V, na zunaj jim je podoben, a znatno spremenjen navznoter (zmogljivosti so uglašene za mikroračunalnike).

Dve zelo dobri knjigi opisujeta notranjo sestavo dveh glavnih vrst. Ti knjigi sta:

- System V: „*Design of the Unix Operating System*“, M. J. Bach.
- BSD: „*Design and Implementation of the 4.3BSD Unix Operating System*“, Leffler, McKusick, Karels, Quaterman

Za dober uvod v OSF/1 (ne tako tehničen kot prejšnji dve knjigi), glejte: „*Guide to OSF/1, A Technical Synopsis*“, izdano pri založbi O'Reilly. Na SunOS, „*Virtual Memory Architecture in SunOS*“ in „*Shared Libraries in SunOS*“ v spisih „*USENIX Proceedings*“, poletje 1989.

Dober nabor člankov o prihodnosti Unixa je „*Unix Variants*“ v izdaji revije Unix Review, april 1992. Drugi dobri viri informacij vključujejo datoteko `bsd-faq` in mnoge novičarske skupine, omenjene v besedilu.

7.4 Glavni igralci in standardi Unixa

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Mon Jan 21 16:59:14 EST 1995
X-Version: 2.8
```

Najpomembnejši igralci na odru Unixa so trenutno (zgodnje leto 1995, popravki so zelo dobrodošli, to so novi byti):

- Novell, ki je kupil USL (zgodaj 1993) in ima zdaj izvorno kodo.
- X/Open, ki ima tržne pravice do uporabe blagovne znamke „UNIX“.
- OSF, kot razvijalec OSF/1 in Motif, in kot organizacija, ki vodi projekt CODE (OSF-jev nov delovni model). Organizacija OSF je bila reorganizirana leta 1994 (in Sun se je priključil), odnosi z X/Open so bili formalizirani.
- IEEE s standardi POSIX, LAN.
- PowerOpen (IBM, Apple, Motorola, Bull, drugi), ki promovira PowerPC. Ne zamešajte ga z grafičnim okoljem z istim imenom.

Naslednje kratko opisuje najpomembnejše standarde, ki se nanašajo na Unix.

- IEEE:
  - standardi 802.x (LAN) (LLC, ethernet, token ring, token bus)
  - POSIX (ISO 9945?): Portable Operating System I/F - prenosljiv vmesnik operacijskega sistema (Unix, VMS in OS/2!) (samo ? ti so bili do sedaj končani)
    - 1003.1: knjižnične procedure (večinoma sistemski klici) - približno V7 razen za signale in terminalski vmesnik (1990)
    - 1003.2: ukazna lupina in pripomočki
    - 1003.3: testne metode in ustreznost
    - 1003.4: realni čas: binarni semaforji, zaklepanje pomnilnika procesa, v pomnilnik preslikane datoteke, deljen pomnilnik, razporejanje prioritet, signali v realnem času, ure in časomerilci, podajanje sporočil IPC, sinhroniziran V/I, asinhron V/I, datoteke v realnem času
    - 1003.5: povezave jezika Ada

        * 1003.6: varnost
        * 1003.7: sistemsko upravljanje (vključuje tiskanje)
        * 1003.8: transparenten dostop do datotek
        * 1003.9: povezave jezika FORTRAN
        * 1003.10: super-računalništvo
        * 1003.12: vmesniki, neodvisni od protokola
        * 1003.13: profili v realnem času (angl. real-time profiles)
        * 1003.15: vmesniki za super-računalniški batch (?)
        * 1003.16: povezave jezika C (?)
        * 1003.17: storitve z imeniki
        * 1003.18: standardiziran profil po POSIX (?)
        * 1003.19: povezave jezika FORTRAN 90

- X/Open (konzorcij proizvajalcev, ustanovljen 1984):
    - X/Open Portability Guides (XPGn) - vodniki po združljivosti z X/Open:
        * XPG2 (1987), močan vpliv SV
            · Vol 1: ukazi in pripomočki
            · Vol 2: sistemski klici in knjižnice
            · Vol 3: terminalski vmesnik (`curses`, `termio`), IPC (SV), internacionalizacija
            · Vol 4: programski jeziki (C, COBOL!)
            · Vol 5: upravljanje s podatki (ISAM, SQL)
        * XPG3 (1989) dodaja: X11 API (sistemski klici za X11)
        * XPG4 (1992) dodaja: XTI? 22 komponent
    - zaporedja vmesnikov XOM:
        * XOM (X/Open Object Management) splošni mehanizmi vmesnikov za naslednje
        * XDS (X/Open Directory Service)
        * XMH (X/Open Mail ??)
        * XMP (X/Open Management Protocols) - ne Bullov CM API?
    - X/Open ima zdaj pravice za uporabo blagovne znamke „UNIX“ (pozno 1993);
    - „Spec 1170“ - ta specifikacija v pripravi opisuje skupen API (programski vmesnik aplikacij), ki mu bodo morali zadostiti proizvajalci, ki želijo uporabljati ime „UNIX“ (ko bodo na voljo zbirke testov). Združitev SVID, OSF-jev AES in druge reči.

- AT&T - (je to še relevantno leta 1994? Kdo je zdaj odgovoren za SVID, TLI, APLI?)
    - System V Interface Definition (SVID) - definicija vmesnika Systema V
        * SVID1 (1985, SVR2)
            · Vol 1: sistemski klici in knjižnice (podobno XPG2.1)
        * SVID2 (1986, SVR3)
            · Vol 1: sistemski klici in knjižnice (osnova, razširitve jedra)
            · Vol 2: ukazi in pripomočki (osnovni, napredni, upravniški, razvoj programja), terminalski vmesnik
            · Vol 3: terminalski vmesnik (spet), STREAMS in TLI, RFS
        * SVID3 (19??, SVR4) dodaja
            · Vol 4: ?? &c
    - vmesniki API (programski vmesnik za aplikacije)

    * Transport Library Interface (TLI)
    * ACSE/Presentation Library Interface (APLI)

- COSE (COmmon Open Software Environment - skupno okolje odprtega programja) [IBM, HP, SunSoft in drugi]: cilj je zbližati različne platforme Unixa. Iniciative v naslednjih področjih:
  - namizna okolja
  - aplikacijski vmesnik API (kot Spec 1180 - en sam programski vmesnik) - verjetneje najpomembnejši dosežek na tej točki; odstrani razlike med SCO, AIX, Solaris, HP-UX, UnixWare.
  - porazdeljene računske storitve (OSF-jev DCE in SunSoftov ONC)
  - objektne tehnologije (OMG-jeva CORBA)
  - grafika
  - večpredstavnost
  - upravljanje sistemov
- PowerOpen Environment (POE, okolje PowerOpen), ki ga promovira združenje PowerOpen (POA). Standard za Unixu podobne OS, ki tečejo na čipu PowerPC. Definira:
  - API (programski vmesnik za aplikacije, izpeljan iz AIX, ustreza POSIX, XPG4, Motif, &c) in
  - ABI (binarni aplikacijski vmesnik), razločljiv faktor med drugimi standardi kot so POSIX, XPG4, &c.. Katerikoli sistem, ki bo ustrezal POE, bo lahko poganjal katerokoli programje za POE.

  Ključne odlike:
  - zasnova na arhitekturi PowerPC
  - neodvisnost od strojnega vodila
  - implementacije sistema se lahko raztezajo od prenosnikov do super-računalnikov
  - potrebuje večuporabniški, večopravilni operacijski sistem
  - omrežna podpora
  - razširitve X windows, Motif
  - ustreznost preizkušena in potrjena od neodvisne stranke (POA)

  AIX 4.1.1 bo ustrezal PowerOpen. MacOS ne ustreza in tudi ne bo. (Zgornje je prirejeno iz `powerpc-faq` novičarske skupine `comp.sys.powerpc`.)

IBM je vpleten v projekt COSE in v projekt POE. Kako neki bosta tadva sodelovala?

7.5 Ugotovitev vrste vašega Unixa

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Mon May 30 15:44:28 EDT 1994
X-Version: 2.6
```

Ta razdelek našteva vrsto stvari, ki jih lahko pogledate v poskusu, da bi ugotovili osnovno vrsto vašega Unixa. Glede na precejšnjo izmenjavo kode in idej med različnimi vrstami in mnogih sprememb, ki so jih naredili proizvajalci, je vsaka izjava oblike „Ta Unix je tipa SVR2¡‘ v najboljšem primeru statistična (razen nekaterih prenosov SVRn). Veliko Unixov ponuja tudi večino iz obeh svetov (pomešano, kot SunOS, ali striktno razločeno, kot Apollo?). Zato ta razdelek verjetno ni preveč uporaben ...

Pomaga vam lahko tudi seznam lastnosti iz prejšnjih razdelkov. Na primer, če ima sistem `poll(2)`, ne pa tudi `select(2)`, je zelo verjetno, da izhaja iz SVR3. Namig vam lahko da tudi ime OS, kot tudi prijavno sporočilo (npr. SGI-jevo „IRIX SVR3.3.2“) ali izhod ukaza „`uname -a`“. Dostopni ukazi vam lahko tudi dajo namig, a to je verjetno manj zanesljivo kot lastnosti jedra. Na primer, tip terminalske inicializacije (inittab ali ttys) je bolj zanesljiv indikator, kot tiskalniški podsistem.

Lastnost	Tipično v SVRx	Tipično v xBSD
ime jedra	/unix	/vmunix
terminalska inicializacija	/etc/inittab	/etc/ttys (le getty do 4.3)
zagonska inicializacija	imeniki /etc/rc.d	datoteke /etc/rc.*
mountani datotečni sistem	/etc/mnttab	/etc/mtab
običajna ukazna lupina	sh, ksh	csh, #! hack
domorodni dat. sistem	S5 (blk: 512-2K) imena dat. <= 14 bytov	UFS (blk: 4K-8K) imena datotek < 255 bytov
skupine	potrebuje newgrp(1) SVR4: več skupin	avtomatsko članstvo
tiskalniški podsistem	lp, lpstat, cancel	lpr, lpq, lprm (lpd daemon) ??
nadzor terminala	termio, terminfo, SVR4: termios (POSIX)	termios (sgtty pred 4.3reno) termcap
nadzor opravil	>= SVR4	da
ukaz ps	ps -ef	ps -aux
večkratni wait	poll	select
string fcns	memset, memcpy	bzero, bcopy
mapiranje procesov	/proc (SVR4)	

Ko se pomikamo v pozna 90-ta leta, je to verjetno vedno manj in manj relevantno.

7.6 Kratki opisi nekaterih znanih (komercialnih/prostih) Unixov

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Tue Aug 15 15:14:00 EDT 1995
X-Version: 2.9
```

(Sploh nisem zadovoljen s tem razdelkom, žal nimam niti časa, niti dokumentacije, da bi ga zelo izboljšal. Vseboval naj bi le Unixe, znane širšemu občinstvu. Majhni in neameriški Unixi so dobrodošli, npr. Eurix. Nujno preformatiranje.)

Ta razdelek našteva (po abecedi) nekaj bolj znanih Unixov skupaj s kratkim opisom njihove narave. Žal je zastarel skoraj po definiciji ...

AIX: IBM-ov Unix, temelji na SVR2 (poznejši na SVR3.2?) z raznimi stopnjami razširitev BSD za različno strojno opremo. Lasten sistemski upravljalnik (SMIT). Podpira CP 850 in Latin-1. Precej različen od večine Unixov in tudi med sabo. Novičarska skupina: `comp.unix.aix`.

- 1.x (za 386 PS/2)
- 2.x (za PC RT)
- 3.x (za RS/6000), *paging kernel*, *logical volume manager*, i18n;
- 3.2 dodaja TLI/STREAMS. Bazira na SV z veliko izboljšavami.
- 4.1 je zadnji (vključuje podporo za PowerPC?)
- AIX/ESA, teče na računalnikih *mainframe* S/370 in S/390, temelji na OSF/1. AIX bi moral biti osnova za OSF/1, a so namesto njega izbrali Mach. Upam, da ta podrazdelek konvergira. `:-)`

AOS (IBM): Prenos 4.3BSD na IBM PC RT (za izobraževalne ustanove). Ne zamešajte ga z DG-ovim OS istega imena.

Arix: SV

A3000UX (Commodore): Unix (?) SVR4 za računalnike Amiga s procesorjem Motorola 68030.

A/UX (Apple): SV z razširitvami z Berkeleya, NFS, Mac GUI. System 6 (pozneje System 7) teče kot gost A/UX (obratno kot MachTen). Novičarska skupina: `comp.unix.aux`.

- 2.0: SVR2 z 4.2BSD, aplikacije za Mac System 6.
- 3.0 (1992): SVR2.2 z 4.3BSD in razširitvami SVR3/4; X11R4, MacX, TCP/IP, NFS, NIS, RPC/XDR, različne ukazne lupine, UFS ali S5FS. aplikacije za System 7.
- 4.0 bo OSF/1. Toda slišim, da se je Apple odločil pustiti A/UX (zdaj bo skupaj z IBM uporabljal AIX za PPC).

3B1 (680x0): Osnovan na SV, naredil Convergent za AT&T. Novičarska skupina: `comp.sys.3b1`.

BNR/2: pomeni BSD Net/2 Release? Vključuje NetBSD/1, FreeBSD.

BOS za Bullov DPX/2 (680x0)

- V1 (1990): SVR3 z razširitvami BSD (FFS, `select`, vtiči), simetrično MP, X11R3
- V2 (1991): dodaja nadzor opravil, zrcaljenje diska, varnost C2, razširitve DCE
- Obstaja tudi BOS/X, in z AIX združljiv Unix za delovne postaje Bull PPC. Ni znano, v kakšnem sorodu je z zgornjima dvema.

386BSD: Jolitzov prenos programja Net/2. POSIX, 32-biten, še vedno v stanju alpha (zdaj različica 0.1).

BSD/386 (80386): od BSDI, z izvorno kodo (naraslo programje Net2) Novičarska skupina: `comp.unix.bsd`.

Chorus/MiXV: Unix SVR3.2 (SVR4) okoli zasnove Chorus, ABI/BCS.

Coherent (Mark Williams Company): Za 80286. klon Unixa, združljiv z V7, nekaj SVR2 (IPC). V4.0 je 32-bitna. Novičarska skupina: `comp.os.coherent`. Mark Williams je ukinil podjetje v zgodnjem 1995.

Consensys: SVR4.2

CTIX: osnova SV, od podjetja Convergent

D-NIX: SV

DC/OSx (Pyramid): SVR4. Novičarska skupina: `comp.sys.pyramid`.

DELL UNIX [DELL Computer Corp.]: SVR4

DomainIX: glejte DomainOS spodaj.

DomainOS (Apollo, zdaj HP): lasten OS; vrhnja plast je BSD4.3 in SVR3 (proces lahko uporabi enega, nobenega ali oba). Razvoj je zdaj ustavljen, nekatere lastnosti so zdaj v OSF/1 (in NT). Zdaj na SR10.4. Ime za SR9.* je bilo DomainIX. Novičarska skupina: `comp.sys.apollo`.

DVIX (NT-jev DVS): SVR2

DYNIX (Sequent): na osnovi 4.2BSD

DYNIX/PTX: na osnovi SVR3

EP/IX (Control Data Corp.): za MIPS 2000/3000/6000/4000; osnova je RISC/OS 4 in 5, zadošča POSIX-ABI. Načini SVR3, SVR4 in BSD.

Esix (80386): čisti SVR4, X11, OpenLook (NeWS), Xview

Eurix (80?86): SVR3.2 (Nemčija)

FreeBSD: 386bsd 0.1 s popravki in veliko osveženimi pripomočki.

FTX: Stratus fault-tolerant OS (stroj 68K ali i860-i960)

Generics UNIX (80386): SVR4.03 (Nemčija)

GNU Hurd (?): „vaporware“ *Free Software Foundation* (FSF): Emulator Unixa na jedru Mach 3.0. Veliko GNU-jevih orodij je zelo popularnih (Emacs) in uporabljanih v prostih Unixih.

HELIOS (Perihelion Software): za transputer INMOS in veliko drugih platform.

HP-UX (HP): starejši iz S III (SVRx), zdaj SVR2 (4.2BSD?) s pripomočki SV (imajo probleme z odločanjem).

- 6.5: SVR2
- 7.0: SVR3.2, symlinks
- 7.5
- 8.0: osnova BSD? za HP-9000 CISC (300/400) in RISC (800/700), delitev knjižnic
- 9.0: vključuje DCE

Interactive SVR3.2 (80x86): čisti SVR3. Interactive je kupil Sun; bo njihov sistem preživel Solaris?

Idris: prvi klon Unixa, naredil ga je Whitesmith. Majhen Unix? Za transputerje INMOS in druge?

IRIX (SGI): Različica 4: SVR3.2, precej BSD. Različica 5.x (trenutna je 5.2) temelji na SVR4. Novičarska skupina: `comp.sys.sgi`.

Linux (386/486/586): Unix pod GNU Public License (vendar ne od FSF). Dostopen z izvorno kodo. Zadošča POSIX z razširitvami SysV in BSD. Prenaša se na Alpha/AXP in PowerPC (prenosi na Amige in Atarije s procesorji 680x0 že obstajajo; prenos se dela tudi na MIPS/4000). Novičarske skupine: `comp.os.linux.`{`admin`,`announce`, `development`,`help`,`misc`}.

MacBSD, ?: deluje na Mac II (neposredno na strojno opremo).

MachTen, Tenon Intersystems: teče kot gost pod MacOS; okolje 4.3BSD s TCP, NFS. Pomanjšana različica: MachTen Personal.

MacMach (Mac II): 4.3BSD na mikrojedru Mach 3.0, X11, Motif, programje GNU, izvorna koda, poskusni System 7 kot opravilo Mach. Popoln z vso izvorno kodo (potrebuje licenco Unix).

Mach386: od Mt Xinu. Temelji na Mach 2.5, z razširitvami 4.3BSD-Tahoe. Tudi 2.6 MSD (Mach Source Distribution).

Microport (80x86): čisti SVR4, X11, OpenLook GUI

Minix (80x86, Atari, Amiga, Mac): klon Unixa, združljiv z V7. Prodaja se z izvorno kodo. Se POSIXizira. Za PC-je in gotovo za veliko drugih (npr. transputer INMOS). Novičarska skupina: `comp.os.minix`.

MipsOS: SV-jevski (RISC/OS, zdaj opuščen, je bil BSD-jevski)

more/BSD (VAX, HP 9000/300): Mt Xinu-ov Unix, temelji na 4.3BSD-Tahoe.

NCR UNIX: SVR4 (4.2?)

Net/2 tape (Berkeley, 1991): BSD Unix, posebej združljiv z 4.3BSD, vključuje le kodo, prosto kode AT&T, nobene nizkonivojske kode. Glejte 386BSD in BSD/386 zgoraj.

NetBSD 0.8: je pravzaprav 386bsd v novi obleki. Prenesen na računalnike [34]86, MIPS, Amiga, Sun, Mac. Kakšno je razmerje z Net/2?

- 1.0 je prišla ven v 1994

NEXTSTEP (Intel Pentium in 86486, Hewlett-Packard PA-RISC, NeXT 68040): BSD4.3 na jedru Mach, lastni grafični vmesnik.

- 1.x, 2.0, 2.1, 2.2, 3.0, 3.1 (stare)
- 3.2 (trenutna različica, Intel Pentium in 86486, Hewlett-Packard PA-RISC, NeXT 68040)
- 3.3 (prihaja; na voljo različica za SPARC)
- 4.0 (bo napovedana, vključevala bo različico za Sun SPARC in bo upoštevala OpenStep)
- NEXTSTEP za PowerPC ali DEC Alpha še ni napovedan (ali ga načrtujejo?)

NEWS-OS (Sony)

- 3.2

OSF/1 (DEC): DEC-ov prenos OSF/1. Mislim, da je to zdaj (april 1993) dostopno na DEC-ovi zadnji 64-bitni Alphi AXP.

OSx (Pyramid): Dvojni prenos SysV.3 in BSD4.3. Novičarska skupina: `comp.sys.pyramid`.

PC-IX (IBM 8086): SV

Plan 9 (AT&T): napovedan 1992, popoln ponoven napis, ni jasno, kako blizu Unixa je. Ključne točke: distribuiran, zelo majhen, različni stroji (Sun, Mips, Next, SGI, generic hobbit, 680x0, PC-ji), C (ne C++, kot pravijo govorice), nov prevajalnik, okenski sistem „8 1/2" (tudi zelo majhen), 16-bitni Unicode, CPE/datotečni strežnik na zelo hitrih omrežjih.

SCO Xenix (80x86): Različice za XT (nerobustne!), 286, 386 (s preklapljanjem pomnilnika na zahtevo (angl. demand paging)). Danes je večino kode iz Systema V. Stabilen izdelek.

SCO Unix (80x86): SVR3.2 (prenehal jemati kodo USL na tej točki).

Sinix [Siemens]: osnova System V.

Solaris (Sparc, x86):

- 1.0: pravzaprav isti kot SunOS 4.1.1, z OpenWindows 2.0 in pripomočki DeskSet.
- 1.0.1: SunOS 4.1.2 z multiprocesiranjem (jedro ni večnitno); ni za 386
- 2.0: (sprva napovedovan kot SunOS 5.0 v 1988) počiva na SVR4 (s simetričnim MP?), vključeval bo podporo za 386; z OpenWindows 3.0 (X11R4) in OpenLook, DeskSet, ONC, NIS. Oba formata binarnih datotek: a.out (BSD) in elf (SVR4). Podpora za Kerberos. Prevajalniki odvezani!
- Solaris zadošča OpenStep (ne-NeXT, toda z NEXTSTEP API) z zadnjo različico (1994?).
- Sun bo poslal svojo implementacijo OpenStep s projektom DOE za Solaris. Prve različice bodo za Sunove stroje SPARC, toda pozneje se bo pojavila različica za Solaris 2.4 za x86 in PowerPC.

UHC (80x86): čisti SVR4, X11, Motif

Ultrix (DEC): temelji na 4.2BSD z veliko 4.3. Novičarska skupina: `comp.unix.ultrix`.

- 4.4 je zadnja

UNICOS (Cray): osnova System V. Novičarska skupina: `comp.unix.cray`.

- 5.x, 6,x, 7.0

UnixWare Release 4.2 [Univel]: SVR4.2; čez NetWare. Univel ne obstaja več.

UTEK (Tektronix)

- 4.0

VOLVIX (Archipel S.A.): OS na osnovi Unix, zgrajen okoli mikro-jedra temelječega na komunikaciji, distribuciji in realnem času. Sistemski klici SVR3.2, BSD4.4 datotečni/omrežni sistemski klici (VFS, FFS). Tudi NFS in X11. Pravi VOLVIX je za transputerje.

Xenix (80x86): Prvi Unix na Intelovi strojni opremi, temelji na SVR2 (prej na S III in celo V7). Novičarska skupina: `comp.unix.xenix`.

7.7 Unixi, delujoči v realnem času

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Tue Aug 15 15:14:00 EDT 1995
X-Version: 2.9
```

Pozor! Ta razdelek hudo potrebuje dodatno delo. Je poln napak in je nepopoln. Upam, da ga bom imel čas pogledati to zimo (bilo je „to jesen"). Dvomim, da so vsi spodnji res Unixi - prispevki so dobrodošli. Seznam tudi vključuje pogostejše Unixe z lastnostmi realnega časa, in nekatere ne-Unixaške sisteme z Unixu podobnim API. Mislim, da zadnji ne spadajo zares sem, a ko sem že zbral nekaj podatkov jih nočem zavreči. Glejte tudi novičarsko skupino `comp.realtime`.

AIX: AIX/6000 ima podporo realnemu času.

Concurrent OS (Concurrent): pravi Unix, ki ga je Concurrent znatno spremenil.

CX/UX: pravi UNIX, ki ga je znatno spremenil Harris, da bi ponujal zmožnosti realnega časa. Zadošča končni različici POSIX.4.

EP/LX (Control Data): prenos sistema LynxOS na R3000. Prej se je imenoval TC/IX.

LynxOS (Lynx Real-Time Systems, Inc): združljiv z Berkeley in SV, popoln ponoven napis (v lasti podjetja), prednik SVR5. Ni UNIX, a podpira večino vmesnikov Unixa (SV in BSD). Upošteva POSIX. Popolnoma predkupen (angl. preemptive), fiksne prioritete.

MiX: implementacija SVR4 z mikrojedrom, izdelal Chorus.

Motorola SVR4 ima zmožnosti realnega časa.

QNX (Quantum Software): realno časovni OS, združljiv z Unixom.

REAL/IX: temelji na Systemu V 3.2 z odlikami RT features (povsem predkupno jedro, razporejevalnik fiksnih prioritet, časomerilec RT, &c.). Za sisteme, grajene na 68xxx in 88xxx. Ustreza POSIX (1003.1 - 1988) in v obliki za 88k ustreza tudi 88open BCS. Dostopen tudi za x86/Pentium.

RTMX O/S [RTMX Incorporated]: elementi NET2, 4.4BSD-Lite in lastne kode. Vključuje tudi orodja FSF. Razširitve realnega časa (POSIX).

RTU (Concurrent), za škatle z 68K

Solaris 2 ima realno časovne zmožnosti?

Stellix (Stardent); je Unix, ali je tudi realno časovni?

Venix/386: Interactive SVR3.2 z realno časovnimi razširitvami.

VMEexec (Motorola): ni Unix, a deli nekatere vmesnike z Unixom.

VxWorks (Wind River Systems): Malo skupnega z Unixom, le nekateri vmesniki (a ne datotečni sistem). Novičarska skupina: `comp.os.vxworks`.

(o naslednjih ne vem ničesar)

Convex RTS

REAL/IX (AEG)

Sorix (Siemens)

System V/86 (Motorola)

TC/IX (CCD)

Velocity (Ready Systems)

7.8 Slovarček Unixa

```
From: "Pierre (P.) Lewis" <lew@bnr.ca>
Date: Tue Aug 15 15:14:00 EDT 1995
X-Version: 2.9
```

Ta razdelek ponuja kratke definicije različnih konceptov in komponent sistemov Unix (ali: povezanih s sistemi Unix).

Chorus: mikrojedro s prenosom sporočil (angl. message-passing microkernel), lahko predstavlja osnovo za prihodnje izdaje SV. Na Chorusu že tečejo SVR4 (binarno združljivi).

CORBA (Common Object Request Broker Architecture).

COSE (Common Open Software Environment) [Sun, HP, IBM]: skupni videz in občutek (Motif - Sun bo pustil Open-Look odmreti), skupni API. Reakcija proti Windows NT. Glejte zgornji razdelek 7.4 („Glavni igralci in standardi Unixa“).

DCE (Distributed Computing Environment, od OSF): Vključuje RPC (NCS Apolla), imeniške storitve (lokalne na DNS, globalne na X.500), časovne, varnostne in nitne storitve, DFS (distribuirani datotečni sistem), neodvisen od OS.

DME (Distributed Management Environment, od OSF): prihodnost.

DO (Distributed Objects [Enterprise]): ???.

FFS (Fast File System): od Berkeley, 1983. Ekvivalent (točen?) UFS v SunOS. Ima pojme kot so skupine cilindrov, fragmenti.

FSF (Free Software Foundation)

Mach: sodobna jedra z univerze CMU (Carnegie Mellon University), na katerih temelji veliko Unixov in drugih OS (npr. OSF/1, MacMach, ...):

- 2.5: monolitno jedro z 4.2BSD
- 3.0: mikrojedro s strežnikom BSD Unix v uporabniškem prostoru (in drugi OS, npr. MS-DOS)

Novičarska skupina: `comp.os.mach`.

MFS (Memory File System): pomnilniški datotečni sistem

NeWS (Network extensible Window System), podjetja Sun?: temelji na PostScriptu, omrežen, zbirke orodij (in tudi odjemnikov) naloženih v strežnik. Del OpenWindows.

NFS (Network File System): Sun je prispeval v BSD, strežnik brez stanj (angl. stateless server)

ONC (Open Network Computing): Sun(?), vključuje RPC, imenski strežnik (NIS, znan tudi kot YP), NFS, ... (najdemo ga v veliko Unixih in drugih OS).

OpenStep [NeXT, Sun]: ???

PowerOpen: standard in organizacija za promocijo PowerPC. Vključuje IBM, Apple in Motorolo; drugi? Glejte zgornji razdelek 7.4 („Glavni igralci in standardi Unixa“).

PowerPC (PPC): procesorski čip RISC [IBM, Motorola].

RFS (Remote File System): SV, strežnik s stanji (angl. stateful server), nezdružljiv z NFS

RPC (Remote Procedure Call): visoko-nivojski mehanizem IPC (med-procesna komunikacija). Dve vrsti.

- ONC: Čez TCP ali UDP (pozneje OSI), uporablja XDR za šifriranje podatkov.
- DCE: ima drugačen mehanizem RPC (temelji na NCS Apolla)

S5 FS: domorodni datotečni sistem Systema V, bloki od 512 bytov do 2 Kb.

sockets (vtiči): BSD-jev vmesniški mehanizem za omrežja (primerjajte s TLI).

STREAMS: mehanizem podajanja sporočil, najprej v SVR3, ki poskrbi za zelo dober vmesnik za razvoj protokolov.

TFS (Translucent File System): Sun, COW, uporabljen na datotekah.

TLI (Transport Library Interface): vmesnik SV do transportnih storitev (TCP, OSI). UI je tudi definiral APLI (ACSE/Presentation Library Interface)

UFS (?): domorodni sistem BSD, kot ga vidi SunOS, bloki od 4 Kb do 8 Kb, skupine cilindrov, fragmenti.

XTI (X/Open Transport Interface): TLI z izboljšavami

X11: okenski sistem z MIT-a, orientiran na pike

8 Primerjava sistemov za upravljanje konfiguracij (RCS, SCCS).

8.1 RCS vs. SCCS: Uvod

```
Subject: RCS vs SCCS:  Introduction
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

V nedavni anketi je bila večina glasov v prid RCS, nekaj v prid SCCS, in nekaj predlaganih alternativ, kot je CVS.

Funkcionalno sta RCS in SCCS praktično ekvivalentna. RCS ima nekaj več odlik, saj se še naprej osvežuje.

Vedite tudi, da se je RCS učil na napakah SCCS ...

8.2 RCS vs. SCCS: Kako sta primerljiva vmesnika?

```
Subject: RCS vs SCCS:  How do the interfaces compare?
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

RCS ima za nove uporabnike preprostejši vmesnik. Obstaja manj ukazov, je bolj intuitiven in konsistenten, in ponuja več uporabnih argumentov.

V SCCS morajo biti veje posebej ustvarjene. V RCS se preverjajo kot katerikoli druga različica.

8.3 RCS vs. SCCS: Kaj je v datoteki popravkov?

```
Subject: RCS vs SCCS:  What's in a Revision File?
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

RCS shranjuje zgodovino v datotekah s podaljškom „`,v`“. SCCS shranjuje zgodovino v datotekah s predpono „`s.`“.

RCS v trenutnem imeniku in podimeniku za RCS samodejno poišče datoteke za RCS, lahko pa določite tudi kakšno drugo datoteko za RCS. Uporabniški vmesnik `sccs` programa SCCS vedno uporablja imenik za SCCS. Če ne uporabljate uporabniškega vmesnika `sccs`, morate določiti polno ime datoteke za SCCS.

RCS shranjuje svoje popravke tako, da vzdržuje kopijo zadnje različice in razlike besedila do prejšnjih različic. SCCS uporablja koncept „združevanja razlik“.

Vsa aktivnost RCS se dogaja v eni sami datoteki za RCS. SCCS vzdržuje več datotek. To je lahko neredno in begajoče.

Ročno urejanje datotek za RCS ali za SCCS je slaba zamisel, saj so napake tako lahke za storiti in tako usodne za zgodovino datoteke. V obeh primerih je lahko urejati podatke o popravkih, vendar ne želite popravljati resničnega besedila različice v RCS. Če urejate datoteko za SCCS, boste morali preračunati kontrolno vsoto s programom `admin`.

8.4 RCS vs. SCCS: Katere so ključne besede?

```
Subject: RCS vs SCCS:  What are the keywords?
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

RCS in SCCS uporabljata različne ključne besede, ki se v besedilu razvijejo. Pri SCCS se, ob izvlečenju datoteke za branje, ključna beseda „`%I%`“ nadomesti s številko popravka.

Ključne besede RCS so lažje za zapomnitev, toda razvijanje ključnih besed se lažje nastavi v SCCS.

V SCCS se ključne besede razvijejo pri izvlečenju le za branje. Če se različica z razvitimi ključnimi besedami prepiše v datoteko, ki bi morala biti razlikovana (angl. deltaed), so ključne besede izgubljene in podatki o različicah datoteke ne bodo osveževani. Po drugi strani, RCS ob razvitju obdrži ključne besede, torej se temu izogne.

8.5 Kaj je simbolično ime v RCS?

```
Subject: What's an RCS symbolic name?
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

RCS vam dovoljuje obravnavati množico datotek kot družino datotek, medtem, ko je SCCS mišljen prvotno za hranjenje zgodovine popravkov datotek.

RCS to omogoča z uporabo simboličnih imen: vse izvorne datoteke, ki so povezane z aplikacijsko različico, lahko označite z uporabo

```
rcs -n
```

in jih pozneje zlahka izvlečete kot združeno enoto. V SCCS bi morali za to napisati skript, ki bi vpisoval in bral vsa imena datotek in različic v in iz datoteke.

8.6 RCS vs. SCCS: Kaj pa hitrost?

```
Subject: RCS vs SCCS:  How do they compare for performance?
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

Ker sistem RCS shranjuje celotno zadnjo različico, je precej hitrejši pri izvlečenju zadnje različice. Od RCS 5.6 naprej je tudi hitrejši od SCCS pri izvlečenju starejših različic.

8.7 RCS vs. SCCS: Identifikacija različic

```
Subject: RCS vs SCCS:  Version Identification.
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

SCCS zna ugotoviti, kdaj se je v sistem dodala določena vrstica kode.

8.8 RCS vs. SCCS: Kako se obnašata v težavah?

```
Subject: RCS vs SCCS:  How do they handle problems?
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

Če vam manjkajo orodja za SCCS ali RCS, ali, če je datoteka popravkov v RCS ali SCCS pokvarjena in orodja nočejo delati z njo, lahko še vedno obnovite zadnjo različico v sistemu RCS. To za SCCS ni res.

8.9 RCS vs. SCCS: Kako se razumeta z ukazom `make(1)`?

```
Subject: RCS vs SCCS:  How do they interact with make(1)?
Date: Wed, 30 Dec 1992 10:41:51 -0700
From: Blair P. Houghton <bhoughto@sedona.intel.com>
```

Dejstvo, da SCCS uporablja predpone za imena datotek (`s.datoteka.c`) pomeni, da jih `make(1)` ne more obravnavati na običajen način, in morate uporabiti posebna pravila (z uporabo znakov ‚~‘), da ga pripravite k sodelovanju s SCCS; kljub temu `make(1)` na nekaterih platformah Unixa ne bo uporabil privzetih pravil za datoteke, ki jih upravlja SCCS.

Zapis s podaljškom (`datoteka.c,v`), ki ga uporablja RCS, pomeni, da lahko v vseh izvedbah programa `make` uporabljate običajna podaljškovna pravila, tudi, če izvedba ni načrtovana posebej za obvladovanje datotek RCS.

8.10 RCS vs. SCCS: Pretvorba

```
Subject: RCS vs SCCS:  Conversion.
Date: Tue, 10 Jan 1995 21:01:41 -0500
From: Ed Ravin <elr@wp.prodigy.com>
```

Za pretvorbo iz SCCS v RCS je na voljo nepodprti skript za ukazno lupino `csh`. Najdete ga lahko na naslovu `<ftp://ftp.std.com/src/gnu/cvs-1.3/contrib/sccs2rcs>`.

Za pretvorbo iz RCS v SCCS morate napisati svoj skript ali program.

8.11 RCS vs. SCCS: Podpora

```
Subject: RCS vs SCCS:  Support
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

SCCS podpira podjetje AT&T. RCS podpira ustanova Free Software Foundation. Torej RCS teče na veliko več platformah, vključno z osebnimi računalniki.

Večina programov `make` prepozna SCCS-jevo predpono „`s.`“, medtem, ko je GNU-jev `make` eden redkih, ki obravnavajo RCS-jev podaljšek „`,v`“.

Nekateri programi `tar` imajo izbiro `-F`, ki ignorira imenike za RCS, za SCCS, ali oboje.

8.12 RCS vs. SCCS: Primerjava ukazov

```
Subject: RCS vs SCCS:  Command Comparison
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

```
SCCS                       RCS                   Razlaga
====                       ===                   =======

sccs admin -i -ndat dat    ci dat                Prvič vpiše datoteko
                                                 ,dat‘ in ustvari
                                                 datoteko sprememb.

sccs get dat               co dat                Izvleče datoteko
                                                 ,dat‘ za branje.

sccs edit dat              co -l dat             Izvleče ,dat‘ za
                                                 spremembo.

sccs delta dat             ci dat                Izvleče predhodno
                                                 zaklenjeno datoteko.

what dat                   ident dat             Izpiše informacijo
                                                 o ključni besedi.

sccs prs dat               rlog dat              Izpiše zgodovino
                                                 datoteke.

sccs sccsdiff -rx -ry dat  rcsdiff -rx -ry dat   Primerja dva popravka.

sccs diffs dat             rcsdiff dat           Primerja trenutno
                                                 različico z zadnjim
                                                 popravkom.

sccs edit -ix-y dat        rcsmerge -rx-y dat    Združi spremembe med
                                                 dvema različicama
                                                 v datoteko.

???                        rcs -l dat            Zaklene zadnji
                                                 popravek.

???                        rcs -u dat            Odklene zadnji
                                                 popravek.  Morda
                                                 stre ključavnico
                                                 drugega uporabnika,
                                                 a mu pošlje
                                                 sporočilo z razlago.
```

8.13 RCS vs. SCCS: Priznanja

```
Subject: RCS vs SCCS:  Acknowledgements
Date: Sat, 10 Oct 92 19:34:39 +0200
From: Bill Wohler <wohler@newt.com>
```

Želim se zahvaliti naslednjim ljudem, ki so prispevali te članke. Na ta spisek želim dodati tudi vaše ime - prosim, pošljite pripombe ali dodatne vire Billu Wohlerju <wohler@newt.com>.

- Karl Vogel <vogel@c-17igp.wpafb.af.mil>
- Mark Runyan <runyan@hpcuhc.cup.hp.com>
- Paul Eggert <eggert@twinsun.com>
- Greg Henderson <henders@infonode.ingr.com>
- Dave Goldberg <dsg@mbunix.mitre.org>
- Rob Kurver <rob@pact.nl>
- Raymond Chen <rjc@math.princeton.edu>
- Dwight <dwight@s1.gov>

8.14 Ali lahko dobim več podatkov o sistemih za upravljanje konfiguracij?

```
Subject: Can I get more information on configuration management systems?
Date: Thu Oct 15 10:27:47 EDT 1992
From: Ted Timar <tmatimar@isgtec.com>
```

Bill Wohler, ki je zbral vse podatke v tem delu teh PZV, je zbral še dosti več podatkov. Dobite jih na naslovu <ftp://ftp.wg.omron.co.jp/pub/unix-faq/docs/rev-ctl-sys>.