Další Předchozí Obsah

3. Základy Bourne shellu

Shell je prostředek komunikace uživatele s operačním systémem. Historicky prvním a dodnes nejdůležitějším je Bourne shell, pojmenovaný po svém autorovi A. S. Bourne, který jej navrhl a naprogramoval na přelomu 60. a 70. let. Principy Bourne shell přejaly všechny následující příkazové interprety, obvykle jde pouze o jiný uživatelský přístup.

V shellu mají jisté znaky speciální význam. Pro rychlou orientaci v těchto znacích je připravena tabulka.

Speciální znaky	Význam
|	roura (pipe) - řazení příkazů do kolony
&	spuštění na pozadí
;	ukončení příkazu
( )	spuštění v kopii shellu
>	presměrováná standardního výstupu
<	presměrování standardního vstupu
&&	spuštění při nulovém návratovem kódu
||	spuštění při nenulovém návratovem kódu
$	umožňuje přístup k promenným
* ? [ ]	znaky pro expanzi jmen
\	potlačuje význam následujícího speciálního znaku
' '	potlačuje význam speciálních znaků
" "	potlačuje význam speciálních znaků až na \, $ a ` `
` `	vrátí standardní výstup příkazu

Některé speciální znaky si ukážeme na příkladech

```
who | wc -l
```
vypíše počet přihlášených uživatelů

```
sh -c 'sleep 5;date'& echo Date:
```
vypíše Date: a po 5 sekundách datum a cas

```
cd;pwd;(cd /etc;pwd);pwd
```
vypíše název domovského adresáře, pak `/etc` a pak zase domovský adresář

```
tr ay ya <in | tee out
```
v souboru `in` prohodí znaky `a` a `y` a výsledek vypíše na obrazovku a do souboru `out`

```
read DIR; test -d $DIR && echo $DIR je adresar
```
zjistí, zda `$DIR` je adresář

```
tar cvf - $*$.[ch] | compress >etc.tar.Z
```
zabalí a zkompresuje soubory v aktuálním adresáři končící na `.c` a `.h`

```
cat a b c > d
```
spojí soubory `a`, `b`, `c` do souboru `d`

```
du -s $HOME
```
ukáže, kolik místa zabírají naše data

```
find /etc -name '*' -exec grep -l xdobiasl {} \;
```
vypíše soubory v adresáři `/etc`, které obsahují dané slovo (vypíše asi `/etc/passwd`)

```
more `which startx`
```
vypíše soubor, kterým se spouští X Window

```
for i in *; do echo "Balim $i"; compress $i; done
```
v aktualnim adresáři zkompresuje každý soubor zvlášť

```
expr `cat /etc/passwd | sort -t : -n -k 3 | tail -1 | awk '{ FS=":"; print $3 }'` + 1
```
vypíše UID o jednicku vyšší, než je v systému nejvyšší

3.1 Přesměrování vstupu a výstupu

Většina programů, která kdy v UNIXu byla a bude napsána, má interaktivní charakter. Rozumíme tím, že program čte vstup z klávesnice uživatelova terminálu a výsledky své činnosti vypisuje na jeho obrazovku.

Každý proces v UNIXu má k dispozici parametrem jádra stanovený počet komunikačních kanálů (50-60) pro manipulaci se soubory. Otevření souboru znamená alokaci kanálu, obsazení další položky v tabulce. Nově vytvořený proces přebírá ("dědí") po svém otci mimo jiné také tuto tabulku otevřených kanálů. Tabulka kanálů obsahuje 3 záznamy i v případě, že otec žádný ze souborů explicitně neotevřel. Jsou to obsazená místa kanálů č. 0, 1 a 2.

Kanál č. 0 je označován jako standardní vstup a je otevřen pro čtení z klávesnice terminálu, kanál č. 1 je standardní výstup, je otevřen pro zápis na obrazovku terminálu a stejně tak je pro zápis na obrazovku terminálu otevřen kanál č. 2, označovaný jako standardní chybový výstup. V rámci shellu je možné manipulovat se standardním vstupem i výstupem za pomoci speciálních znaků. Příkaz

```
$ ls -l > seznam
```

nevypisuje atributy souborů pracovního adresáře na obrazovku terminálu, výpis je ukládán do souboru `seznam`. Je toho dosaženo zápisem speciálního znaku `>`, který shell rozpozná a interpretuje jako přepnutí standardního výstupu procesu `ls` do souboru `seznam`. Soubor `seznam` je nově vytvořen, pokud již existoval, jeho předchozí obsah je ztracen. Standardní výstup lze přesměrovat i způsobem, kdy původní obsah souboru bude zachován, přepnutý obsah standardního výstupu k němu bude pouze připojen. Toho se dosáhne zápisem `>>`:

```
$ ls -l >> seznam
```

Znakem `<` přepínáme standardní vstup.

```
$ rm -i > soubor < potvrzeni
```

čte program `rm` ze souboru `potvrzeni` namísto z klávesnice terminálu.

Standardní chybový výstup se přesměruje do souboru `chyby` takto

```
$ cc prog.c 2>chyby
```

Lze také obecně spojovat tok dat určitého kanálu s kanálem jiným. Např. standardní chybový výstup lze napojit na kanál č. 1 zápisem

```
$ time ls > mereni 2>&1
```

3.1 Roura

Roura je prostředek pro spojení dat dvou procesů. Proces PRODUCENT zapisuje data na standardní výstup a proces KONZUMENT data čte ze standardního vstupu. Znakem `|` vytváří shell rouru. Tímto mechanizmem, který jádro realizuje na požádání shellu, lze propojovat sekvenčně současně několik procesů. Příkazový řádek pak nazýváme *kolona*. Následující příklad na standardní výstup vypisuje počet

souborů v daném adresáři:

```
$ ls | wc -l
```

Užitečný je program `tee`. Jeho název je odvozen z vodoinstalační terminologie. Jde o odbočku z roury. Odbočka svede standardní vstup do souboru daného parametrem a současně jej předává na standardní výstup:

```
$ who | tee kdo | grep jan |  wc -l
```

3.3 Expanzní znaky jmen souborů

V Unixu téměř všechny znaky (s výjimkou písmen abecedy a číslic) mají speciální význam.

`*`	libovolná posloupnost znaků (i nulové délky), nezastupuje řetězec znaků začínající tečkou
`?`	právě jeden libovolný znak
`[znaky]`	právě jeden znak z množiny znaků uvedených v těchto závorkách, při specifikaci lze použít následující dva speciální znaky:
`-`	používa se pro označení intervalů znaků z množiny ASCII (např. a-z, 0-9),
`!`	negace výčtu znaků

Metaznaky můžeme použít pro specifikaci jména souboru v libovolném příkazu. Substituci konkrétních jmen souborů provádí shell, ne vlastní program. Napr. po zadání příkazu

```
$ ls -l /bin/l*
```

shell interpretuje metaznaky, a spustí program ls s příkazovou řádkou:

```
$ ls -l /bin/ld /bin/line /bin/ln /bin/login /bin/logname /bin/ls /bin/ltf
```

3.4 Další znaky zvláštního významu

Sekvenci příkazů lze psát v rámci jednoho vstupního řádku. Znakem `;` oddělujeme příkazy:

```
$ pwd; ls -l
```

Sekvence příkazů můžeme sdružovat do skupin. Závorky `( a )` vymezují skupinu příkazů, je na ni ale implicitně startován nový proces sh.

```
$ ( cat /usr/jan/seznam; ls;) | grep *.c
```

Znaky výluky

`\`	snižuje význam znaku za ním následující na úroveň ostatních znaků, shell neuvažuje jeho speciální význam. Pokud se obrácené lomítko zapíše na konec řádky, potlačí se

	obvyklý význam znaku konce řádky - vznikne pokračovací řádek.
`'řetězec'`	apostrofy označují řetězec, kde znaky zvláštního významu jsou považovány za běžné znaky,
`"řetězec"`	uvozovky označují řetězec, kde znaky zvláštního významu jsou považovány za běžné znaky, kromě znaku dolar ($) a opačné apostrofy (` `),
`$proměnná`	odkaz na obsah příslušné promenne,
`` `příkaz` ``	obrácené apostrofy označují, že se na jejich místě má doplnit standardní výstup programu uvedeného v apostrofech.

3.5 Práce s úlohami

Všechny příkazy, které jsme prozatím v této kapitole uváděli, byly spouštěny na popředí (foreground). Znamená to, že příkaz byl interpretován a uživatel očekával výpis výzvy `$`, kterou mu bylo sděleno, že interpretace byla ukončena. Bez toho, abychom čekali na ukončení příkazu a bylo nám umožněno zadávat další příkazy, je možné spustit příkaz na pozadí (background). Dosáhne se toho speciálním znakem `&`, který bude uveden jako poslední na příkazovém řádku.

```
$ ls -l > seznam &
231
$
```

Na pokyn `$` můžeme ihned zadávat další příkaz. Shell vypsal na standardní výstup identifikační číslo procesu (PID) spuštěného na pozadí, zde `231`. Slouží uživateli k manipulaci s procesem programu na pozadí. Pokud by se např. proces spuštěný na pozadí zacyklil může mu uživatel zaslat signál, na který proces reaguje ukončením činnosti. PID má každý proces v době svého vzniku přiděleno, u procesů běžících na popředí se implicitně nezobrazuje.

Jiný způsob jak dostat proces na pozadí je udělat to, když už program běží. Nejprve spusťte nějaký program. Když běží, stiskněte `control+z`. To ten proces pozastaví. Okamžitě zastaví jeho běh, ale může být kdykoliv znovu rozběhnut. Jakmile je proces pozastaven, vrátí se vám na monitor prompt. Pak můžete rozběhnout pozastavený proces na pozadí tak, že napíšete:

```
$ bg
```

Pokud chcete komunikovat s upozaděným procesem, můžete jej přenést zpět do popředí. Máte-li upozaděný pouze jediný proces, můžete jej přenést do popředí tak, že napíšete:

```
$ fg
```

Je možné mít vícero upozaděných procesů najednou. Pokud se to stane, budete potřebovat vědět jak dostat do popředí jeden konkrétní proces. Pouhým zapsáním `fg` půjde dopředu proces, který byl upozaděn jako poslední. Ale co když máte na pozadí celou řadu procesů? Naštěstí bash obsahuje příkaz, umožňující všechny procesy vypsat. Ten příkaz se jmenuje `jobs` a poskytuje takovýto výstup:

```
$ jobs
[1]   Stopped                 vim
[2]-  Stopped                 amp
[3]+  Stopped                 man ps
```


To vám ukazuje seznam všech procesů, které jsou na pozadí. Jak vidíte, všechny jsou pozastavené (stopped). Číslo na začátku je identifikátorem upozaděného procesu. Číslo za nímž je plus označuje proces, který by se dostal do popředí, kdybyste napsali pouze `fg`.

Pokud chcete dostat do popředí `vim`, musíte napsat:

```
$ fg 1
```

ps

PID všech procesů spuštěných v rámci sezení u terminálu lze získat příkazem `ps`:

```
$ ps
PID   TTY   TIME  COMMAND
31    01    0:03  sh
44    01    0:00  ls
45    01    0:00  ps
$
```

Můžete získat úplný seznam procesů, běžících na vašem systému, když použijete správnou kombinaci voleb. To pravděpodobně vyústí v předlouhý seznam procesů:

```
$ ps -ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:03 init [3]
    2 ?        SW     0:13 [kflushd]
    3 ?        SW     0:14 [kupdate]
    4 ?        SW     0:00 [kpiod]
    5 ?        SW     0:17 [kswapd]
   11 ?        S      0:00 /sbin/kerneld
   30 ?        SW     0:01 [cardmgr]
   50 ?        S      0:00 /sbin/rpc.portmap
   54 ?        S      0:00 /usr/sbin/syslogd
   57 ?        S      0:00 /usr/sbin/klogd -c 3
   59 ?        S      0:00 /usr/sbin/inetd
   61 ?        S      0:04 /usr/local/sbin/sshd
   63 ?        S      0:00 /usr/sbin/rpc.mountd
   65 ?        S      0:00 /usr/sbin/rpc.nfsd
   67 ?        S      0:00 /usr/sbin/crond -l10
   69 ?        S      0:00 /usr/sbin/atd -b 15 -l 1
   77 ?        S      0:00 /usr/sbin/apmd
   79 ?        S      0:01 gpm -m /dev/mouse -t ps2
   94 ?        S      0:00 /usr/sbin/automount /auto file /etc/auto.misc
  106 tty1     S      0:08 -bash
  108 tty3     SW     0:00 [agetty]
  109 tty4     SW     0:00 [agetty]
  110 tty5     SW     0:00 [agetty]
  111 tty6     SW     0:00 [agetty]
  [výstup zkrácen]
```

kill a killall

Občas se stane, že se nějaký proces začne divně chovat, a vy jej potřebujete odstranit. Program, sloužící k takovéto administraci se nazývá `kill` (zabít) a k ovládání procesů může být použit několika způsoby. Nejobvyklejším použitím příkazu `kill` je zabití procesu.

Abyste mohli proces zabít, budete potřebovat znát buď jeho PID, nebo jeho jméno. Pro získání PID použijte příkaz `ps`, který jsme probrali v předchozí sekci. Například, abyste zabili proces s číslem PID 4747, napište tohle:


```
$ kill 4747
```

Upozorňuji, že musíte být vlastníkem procesu, abyste ho mohli zabít. To je bezpečnostní opatření. Kdybyste měli právo zabíjet procesy spuštěné ostatními uživateli, asi by to způsobilo pár zmatků. Nicméně root může zabít jakýkoliv proces v systému.

Ještě máme další variantu příkazu `kill`, nazvanou `killall`. Tento program dělá přesně to, co říká (zab všechny): Zabíjí všechny běžící procesy zadaného jména. Kdybyste chtěli zabít všechny běžící procesy `vim`, napsali byste následující příkaz:

```
$ killall vim
```

Někdy pouhý `kill` na svou práci nestačí. Některé procesy pomocí `kill` prostě nezemřou. Na takové budete potřebovat účinnější formu. Pokud třeba takové urputné PID 4747 neodpovědělo na váš vražedný požadavek, měli byste udělat následující:

```
$ kill -9 4747
```

To téměř jistě přivede proces 4747 k smrti. Totéž můžete udělat pomocí `killall`. Co se přihodilo je to, že byl procesu zaslán jiný signál. Obyčejný `kill` posílá procesům signál SIGTERM (terminate - ukončit). `kill -9` posílá procesům signál SIGKILL (kill - zabít). K dispozici máte celou řadu různých signálů. Jejich seznam můžete získat napsáním:

```
$ kill -l
     1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
     5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
     9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
    13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
    18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
    22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
    26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
    30) SIGPWR
```

top

Na závěr tu máme příkaz, který můžete využít ke zobrazení obnovující se informace o procesech, které v systému běží. Ten příkaz se jmenuje `top` a spouští se takto:

```
$ top
```

Další Předchozí Obsah