Základy operačního systému LINUX – část III

Přesměrování standardního vstupu a výstupu

Přídavná zařízení jsou v systému reprezentována svými řídícími soubory. Řídící soubory umožňují procesům pracovat s přídavnými zařízeními stejně jako s obyčejnými soubory. Proto pro práci s přídavnými zařízeními používají procesy stejné služby systému jako pro práci s obyčejnými soubory.

Předtím než proces začne se souborem pracovat, musí zavolat službu ***open().*** Říkáme, že musí soubor otevřít. Služba *open()* vrátí malé kladné číslo, které se nazývá ***deskriptor souboru***. Pomocí deskriptoru pak proces otevřené soubory identifikuje při volání dalších služeb systému. Z nich jsou nejdůležitější služby ***read()*** pro čtení ze souboru a ***write()*** pro zápis do souboru.

Při otevření souboru proces specifikuje, zda bude ze souboru pouze číst, pak říkáme, že soubor otevřel pouze pro čtení, nebo, zda bude do souboru pouze psát, pak říkáme, že otevřel soubor pouze pro zápis. Existuje ještě třetí možnost a sice, že otevře soubor pro čtení i zápis.

Proces může otevřít vícekrát tentýž soubor. Dokonce jej může otevřít jednou například pro čtení a podruhé pro zápis. Při každém otevření získá jiný deskriptor, takže ze svého hlediska má proces otevřeno několik různých souborů a to třeba i s různými způsoby přístupu.

Při přihlášení uživatele z terminálu, proces **getty** třikrát otevře terminál, na kterém se uživatel přihlašuje. Získá tak tři deskriptory. Protože proces **getty** předtím žádný soubor otevřen neměl, a protože systém přiděluje deskriptory postupně od nuly, získá tak deskriptory 0, 1 a 2. Těmto deskriptorům se říká ***standardní vstup***, ***standardní výstup*** a ***chybový výstup***.

Všechny tyto deskriptory prostřednictvím řídícího souboru odkazují na terminál, na kterém se uživatel přihlásil. Řídící soubory přídavných zařízení jsou v adresáři `/dev` . Řídící soubory terminálů se standardně nazývají `tty1`, `tty2`, . . .

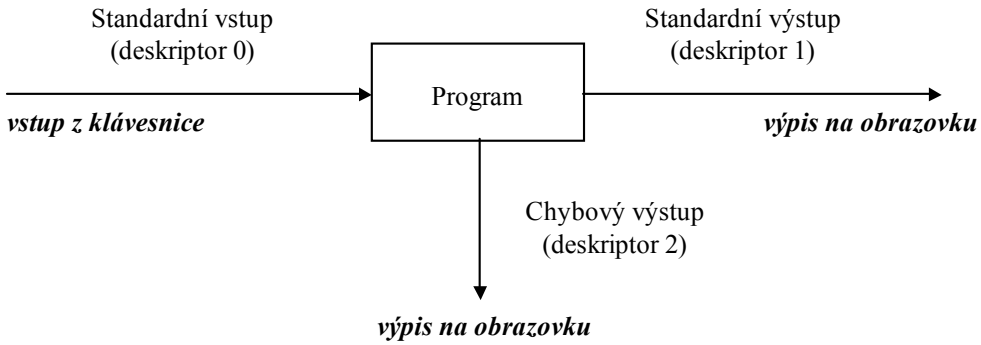
V adresáři `/dev` také najdeme řídící soubory dalších přídavných zařízení, jako jsou disky, disketové jednotky, magnetické pásky atd.

V tomto okamžiku je třeba ještě znát následující řídící soubory. Soubor `/dev/tty` je řídícím souborem tzv. řídícího terminálu procesu. Řídícím terminálem je ten terminál, na kterém se uživatel přihlásil. Tento terminál je za normálních okolností stále přístupný. Je přístupný i tehdy, když byly standardní vstup a standardní i chybový výstup přesměrovány.

Druhým důležitým řídícím souborem je soubor `/dev/null`, který v systému zastupuje funkci koše na odpadky. Pokud zapisujeme do souboru `/dev/null`, nic se nikam nezapisuje, výstup se prostě bezproblémově ztratí.

Po úspěšném přihlášení zdědí login shell deskriptory 0, 1 a 2 a zdědí je i procesy, které jsou z login shellu spuštěny. Pokud tedy shell nebo z něj spuštěné procesy používají tyto deskriptory ke čtení nebo k zápisu, potom čtou nebo píší na terminál.

Programy jsou standardně psány tak, že pokud čtou data z terminálu, používají ke čtení deskriptor 0, pokud vypisují výsledná data na terminál, používají deskriptor 1 a pokud vypisují chybová hlášení, používají deskriptor 2. Jinak řečeno, pokud programy používají ke vstupu a výstupu terminál, potom čtou ze standardního vstupu, výsledky píší na standardní výstup a zprávy o chybách na chybový výstup (viz obr. 6.3).



Obr. Vstup a výstup programu

Shell dovoluje standardní vstup a standardní a chybový výstup přesměrovat.

V Bourne shellu se přesměrování provádí následovně:

`<x`	*Přesměrování standardního vstupu na soubor x*
`>x nebo 1>x`	*Přesměrování standardního výstupu do souboru x*
`2>x`	*Přesměrování chybového výstupu do souboru x*
`2>&1`	*Přesměrování chybového výstupu na standardní výstup*
`1>&2`	*Přesměrování standardního výstupu na chybový výstup*
`>x 2>&1`	Přesměrování standardního a chybového výstupu do souboru `x`

V C shellu jsou možnosti přesměrování omezenější. Jsou možná tato přesměrování:

`<x`	*Přesměrování standardního vstupu do soubor* `x`
`>x`	*Přesměrování standardního výstupu do souboru* `x`
`>&x`	*Přesměrování standardního i chybového výstupu do souboru* `x`

V obou shellech lze znak `>` zdvojit, tj. psát například `>>x`, což znamená přesměrování s připojením. Výstup z programu se nezačne psát od počátku souboru `x` , tj. obsah souboru `x` se nepřepíše, ale výstup z programu se připojí za konec souboru `x`.

Roury

Roury (kolony, pipes) se vytváří pomocí znaku | takto:

```
prog1 | prog2
```

Systém spustí oba programy *prog1* a *prog2* a přesměruje standardní výstup programu *prog1* na standardní vstup programu *prog2*. Pokud chceme přesměrovat spolu se standardním výstupem i chybový výstup, použijeme místo znaku | znak |& . Do roury lze zařadit i scripty, pokud čtou ze standardního vstupu a píší na standardní výstup. Roura může mít i více členů než dva.

Příklad 1:

```
$ls -l /etc | grep "^d" | wc -l
```

Příkaz realizovaný touto rourou spočte počet podadresářů v adresáři `/etc` .

Proměnné a výrazy

V shellu lze stejně jako v programovacích jazycích pracovat s proměnnými.

V shellu především existuje určitý počet předdefinovaných proměnných. Tyto proměnné se obvykle označují velkými písmeny.

Nejdůležitější jsou:

`HOME`	*obsahuje úplnou cestu k domácímu adresáři*
`LOGNAME` (nebo `USR`)	*obsahuje jméno uživatele*
`TERM`	*obsahuje typ terminálu*
`PATH`	*obsahuje seznam adresářů, ve kterých se hledají příkazy*
`PROMPT`	*obsahuje řetězec výzvy (promptu) systému (C shell, implicitně %)*
`PS1`	*obsahuje řetězec promptu (Bourne shell, implicitně* `$` *, pro superuživatele #)*
`PS2`	*řetězec promptu pro pokračování řádku (Bourne shell, implicitně >)*

Hodnoty těchto proměnných může uživatel měnit. Může také definovat další proměnné.

Proměnné jsou dvojího typu:

- ***lokální proměnné*** (***local variables***)
- ***proměnné prostředí*** (***enviromental variables***)

Lokální proměnné se nepřenáší do podřízeného shellu ani nejsou přístupné procesům, které byly shellem spuštěny.

Proměnné prostředí jsou přístupné všem procesům, které shell spustí. Všechny výše uvedené předdefinované proměnné jsou proměnnými prostředí.

Název proměnné je tvořen posloupností alfanumerických znaků a podtržítka. Hodnota proměnné se označuje předřazením znaku **$** před název proměnné.

Příklad 2: Je-li obsah proměnné `USER` novak, pak dostaneme:

```
$echo $USER
novak
```

Rezervované proměnné

Mezi lokální proměnné patří také tak zvané ***reservované proměnné***. Hodnoty těchto proměnných může nastavovat pouze shell. Nejdůležitější reservované proměnné jsou * , # , ? a $. Jejich hodnoty mají následující význam:

`$*`	*všechny parametry příkazového řádku, se kterými byl spuštěn script, který je shellem zpracováván*
`$#`	*počet parametrů příkazového řádku, se kterými byl spuštěn script, který je shellem zpracováván (C shell nemá proměnnou # , počet parametrů příkazového řádku je v proměnné* `#argv`*)*
`$?`	*návratový kód (status ukončení) posledního spuštěného příkazu (C shell nemá proměnnou* `?` *, návratový kód je v proměnné* `status`*)*
`$$`	*PID shellu*

Poziční parametry

Uvnitř scriptu je důležité mít přístup k parametrům spuštění scriptu. K tomu slouží tzv. ***poziční parametry 0,1 ,2, . . . 9***. Jejich hodnotami jsou

`$0`	*název skriptu*
`$1,$2,. . .,$9`	*parametry, se kterými byl skript spuštěn.*

Pokud je příkaz spuštěn s více než devíti parametry, další parametry lze získat pomocí příkazu **shift** . Příkaz **shift** funguje následujícím způsobem:

Poziční parametry jsou uloženy v poli. Přímo přístupných je prvních deset položek tohoto pole (pomocí posičních parametrů *0, 1 , . . . , 9*). Provedení příkazu **shift** způsobí posun obsahu pole o jedno místo vlevo, to jest původní hodnota `$0` je přepsána hodnotou `$1`, hodnota `$1` hodnotou `$2` atd. Zároveň se sníží obsah proměnné # o 1 .

Příklad 3:

```
$cat > prikaz
#!/bin/sh
echo $9
echo $#
shift
echo $9
echo $#
shift
echo $9

$prikaz 1 2 3 4 5 6 7 8 9 10 11 12
9
12
10
11
12
```

Práce s proměnnými se v C shellu a v Bourne shellu podstatně liší. Proto je třeba tuto problematiku vysvětlit pro C shell a Bourne shell zvlášť. Uvedeme zde jen nejdůležitější zásady práce s proměnnými, podrobnosti lze najít v manuálech.

Proměnné a výrazy C shellu

Lokální proměnné mohou obsahovat buď ***slovo (řetězec***) nebo ***seznam slov***. Seznam slov je posloupnost slov, které jsou odděleny mezerami a uzavřeny do kulatých závorek:

Například

`a date data1` *jsou slova*

`(dnes je streda)` *je seznam slov*

Nastavení lokální proměnné se provede následovně (znak | vyjadřuje alternativu):

set *proměnná*=*slovo* | *seznam_slov*

Mezi názvem proměnné a rovnítkem a mezi rovnítkem a hodnotou nesmí být mezera.

Zrušení proměnné se provede příkazem

unset *proměnná*

Výpis nastavení lokálních proměnných se provede příkazem **set** bez parametrů.

Proměnné prostředí mohou obsahovat pouze řetězec znaků. Nastavení proměnné prostředí se provede :

setenv *proměnná řetězec*

Zrušení proměnné se provede příkazem

unsetenv *proměnná*

Výpis se provede příkazem **env** bez parametrů.

Pokud je v proměnné seznam slov, na jednotlivá slova se lze odvolávat jako na prvky pole v jazyce C. Prvky se ale číslují od 1.

Počet slov v proměnné *proměnná* je přístupný prostřednictvím proměnné *#proměnná*.

Příklad 4:

```
%set veta=(dnes je nedele)
%echo $veta[3]
nedele
%echo $#veta
3
%setenv TERM vt100
```

Parametry příkazového řádku jsou v C shellu přístupné dvojím způsobem:

- Jako poziční parametry *0, 1, . . . 9* (další s pomocí příkazu **shift**)
- Jako prvky seznamu slov `argv`: `$1` je totéž co `$argv[1]` , `$2` je totéž co `$argv[2]` atd. Při použití tohoto způsobu označení parametrů není třeba používat příkaz **shift** . Celkový počet parametrů příkazového řádku lze označit jako `#argv` a všechny parametry příkazového řádku jako `argv`.

Aritmeticko-logické výrazy se tvoří pomocí stejných aritmetických a logických operátorů jako v jazyce C. Jako v jazyce C platí:

Pokud je výraz *e* interpretován jako logický, pak platí:

e je pravdivý, pokud $e \neq 0$,

e je nepravdivý, pokud $e = 0$.

Pokud výraz *e* je vyhodnocen jako pravdivý, je mu přiřazena hodnota 1 a pokud je vyhodnocen jako nepravdivý, je mu přiřazena hodnota 0.

Odlišnosti při vytváření výrazů od jazyka C jsou tyto:

- Před hodnotu proměnné je nutné vždy napsat znak `$` .
- Proměnné není třeba předem deklarovat.
- Na obou stranách operátoru musí být mezera.
- Výrazy, ve kterých se vyskytuje některý z operátorů `>` , `<` , `|` , `||` , `&` , `&&` , `^` je nutno uzavřít do závorek, protože jinak by byly shellem interpretovány jiným způsobem (znak > jako přesměrování atd.)
- Ve výrazech mohou vystupovat jen celá čísla a proměnné, jejichž hodnoty jsou znakovým zápisem celého čísla.

Příklad 5: Výrazy jsou například:

```
$a + 10
($a > $b)
```

Přiřazovací příkaz, kterým lze přiřadit hodnotu aritmeticko-logického výrazu proměnné má syntaxi

```
@ proměnná přiřazovací_operátor výraz
```

Přiřazovací operátor může být znak = , += , -= , *= , /= , ++ , -- . Význam těchto operátorů je stejný jako v jazyce C.

Příklad 6:

```
%set  a=7
%set  b=1
%@  v=$a + $b
%echo $v
8
%@ w=($a >> $b)
%echo $w
3
%@  z=($a > $b)
%echo $z
1
%@ z++
%echo $z
2
```

Při provádění přiřazovacího příkazu jsou nejdříve převedeny znakové reprezentace argumentů výrazu na čísla, pak je výraz vyhodnocen a nakonec je výsledek vyhodnocení výrazu opět převeden do znakové reprezentace a uložen do proměnné, která má výsledek vyhodnocení obsahovat.

Proměnné a výrazy Bourne shellu

Proměnné mohou obsahovat pouze řetězce znaků. Proměnná se definuje takto

proměnná=řetězec

Po své definici je proměnná lokální proměnnou. Aby se stala proměnnou prostředí, je třeba zadat její převedení (export) mezi proměnné prostředí. To se provede příkazem

export *proměnná*

Proměnnou prostředí lze také definovat přímo příkazem **export** :

export *proměnná=hodnota*

Proměnnou lze zrušit příkazem **unset**

unset *proměnná*

Výpis nastavení lokálních proměnných lze získat příkazem **set** (bez parametrů). Výpis proměnných prostředí příkazem **env** nebo **export** (bez parametrů).

Příklad 7:

```
$den=nedele
$DATA=regrese.d
$export DATA
$echo $den
nedele
$echo $DATA
regrese.d
```

Proměnnou PATH lze nastavit například takto:

```
$PATH=/bin:/usr/bin:/lib:/usr/lib
$export PATH
```

Proměnnou obsahující prompt lze nastavit tak, aby prompt zobrazoval aktuální adresář (platí pro novější verze Bourne shellu):

```
$PS1=´\w>´
```

V nastavení promptu mohou být ještě následující dvojice znaků:

\t *zobrazení času*

\u *zobrazení jména uživatele*

\h *zobrazení jména počítače*

Shell má zabudovaný vnitřní příkaz **read** pro načtení hodnoty proměnné z klávesnice.

Příklad 8:

```
$read x
100
$echo $x
100
```

Pro vyhodnocování aritmeticko-logických výrazů se používá vnější příkaz (program) **expr**. Ve výrazech místo jmen proměnných vystupují jména jejich hodnot. Hodnoty proměnných musí být celočíselné. Pokud výraz obsahuje operátor, který by vyhodnotil již shell (například >), je třeba tento operátor uzavřít do jednoduchých uvozovek.

Příklad 9:

```
$read a
10
$read b
5
$z=`expr $a ´>´  $b`
$echo $z
15
```

Program **expr** používá následující aritmetické operátory:

+	sčítání
-	odčítání
*	násobení
/	dělení
%	zbytek po dělení

Z relačních operátorů lze použít operátory:

```
<    <=    >    >=    ==    !=
```

Ve výrazech nelze použít závorky. Proto složitější výrazy nutno vyhodnocovat postupně.

Posiční parametry je možné nastavit na libovolné hodnoty příkazem **set** .

Příklad 10:

```
$set prvni druhy treti
$echo $2
druhy
```

Příkaz **set** se používá tehdy, když je třeba z výstupu programu získat určitou položku. Například:

```
$date
Mon Nov 9 14:25:00  GMT+0100 1998
$set `date`
$echo time = $2
time = 14:25:00
```

Mechanismus historie

Pokud je mechanismus historie v činnosti, shell ukládá zadávané příkazy do vyhražené oblasti paměti. Velikost vyhražené paměti lze nastavit. Při nastavení se udává maximální počet naposledy zadaných příkazů, které si bude shell schopen zapamatovat. Tuto hodnotu je třeba uložit do lokální proměnné `HISTSIZE` (platí pro Bourne shell, odpovídající proměnná u C shellu se jmenuje `history`). Pokud tato proměnná není nastavena nebo obsahuje nulu, mechanismus historie není v činnosti.

Příkazy, které shell do vyrovnávací paměti uložil, lze zobrazit vnitřním příkazem shellu **history**.

Na příkazy, které jsou mechanismem historie uloženy se lze odvolávat takto:

!*n*	*n-tý příkaz historie*
!*string*	*poslední příkaz, jehož název začíná řetězcem string*
!!	*poslední příkaz*
!^	*první parametr posledního příkazu*
!$	*poslední parametr posledního příkazu*
!*	*všechny parametry posledního příkazu*
^s1^s2	*poslední příkaz, ve kterém je řetězec s1 zaměněn za řetězec s2*

Příklad 11:

```
$vi prog.c
$cc prog.c -o prog
$prog
$!v                    Provede se vi prog.c
$!c                    Provede se cc prog.c -o prog
$!p                    Provede se prog
```

Nebo:

```
$ls /home/novak/system/pick
$ls -al !*|mose        Provede se
                       ls -al  /home/novak/system/pick|mose
$^os^or                Provede se
                       ls -al  /home/novak/system/pick|more
```

Znaky výluky (Escape characters)

Některé znaky mají v shellu speciální význam. Jakmile na ně shell narazí, interpretuje je podle daných pravidel. Tyto znaky se nazývají ***speciální znaky*** shellu a jsou to například následující znaky:

nový řádek	konec příkazu
mezera	oddělovač příkazu a jednotlivých parametrů
!	odkaz na historii
? * [-] ^	expansní znaky
&	spuštění na pozadí
|	roura
< >	přesměrování

Někdy je třeba zajistit, aby shell speciální znaky neinterpretoval. K tomu slouží ***znaky výluky (escape characters)***.

V shellu jsou k disposici následující *znaky výluky*:

- **Zpětné lomítko (\ )**. Vztahuje se pouze na bezprostředně následující znak. První způsob použití je přepnutí běžného znaku na znak se speciálním významem, např. \n – znak nového řádku, \u – uživatelské jméno, \w – jméno aktuálního pracovního adresáře, \t – tabulátor, \a – spuštění zvonku terminálu, \\ - vlastní znak zpětného lomítka. Druhý způsob použití je vypnutí speciálního významu některého znaku – viz výše, např. `rm *` vymaže všechny soubory ale `rm \*` vymaže pouze soubor s názvem *. Třetí způsob použití dává interpretu najevo, že další řádek je součást jednoho příkazu – znak \ na konci řádku tedy ruší význam odřádkování.
- **Apostrofy (' ').** Vztahují se na všechny znaky, které jsou do nich uzavřeny. Apostrofy vymezují text interpretem chápaný pouze jako text a jako jeden celek. To zahrnuje i zpětná lomítka, znaky nového řádku, jména proměnných, uvozovky, obrácené apostrofy atd.

- **Obrácené apostrofy (` `).** Obrácené apostrofy vsunou výstup v nich uzavřeného příkazu do druhého. Například, příkaz `echo expr 1 + 3 kusy` vypíše text `expr 1 + 3 kusy`, ale příkaz `echo ` expr 1 + 3 ` kusy` vypíše `4 kusy`.
- **Uvozovky (" ")**. Uvozovky vypínají význam expanzních znaků, ale ponechávají příkazy historie, proměnné, obrácené apostrofy a znaky uvozené zpětným lomítkem.

V C shellu platí tato pravidla:

\	Ruší interpretaci všech speciálních znaků.
' '	Ruší interpretaci všech speciálních znaků kromě znaku ! a částečně znaku \ . Znak \ je brán ve speciálním významu pouze v kombinaci **\!** .
" "	Ruší interpretaci všech znaků kromě znaků **!** , **$** , zpětných uvozovek ` ` a částečně znaku \ . Znak \ je brán ve speciálním významu pouze v kombinaci **\!** .

V Bourne shellu platí následující pravidla:

\	Ruší interpretaci všech speciálních znaků.
' '	Ruší interpretaci všech speciálních znaků.
" "	Ruší interpretaci všech znaků kromě **$** a zpětných uvozovek ` ` a částečně i znaku \ . Znak \ je brán ve svém speciálním významu pouze pokud vystupuje v následujících kombinacích **\$** \` \" \\ .

Příklad 12:

```
%echo \*
*
%echo "*"
*
%a=10
%echo 'b=$a'
b=$a
%echo "b=$a"
b=10
%cd \                    znak konec řádku, který následuje po znaku \ , zde nemá význam
                         konec příkazu
/usr/bin
%pwd
/usr/bin
```

Přezdívky a funkce

Příkaz i s parametry spuštění lze pojmenovat novým názvem, tzv. ***přezdívkou*** (***alias***). Příkaz může být jednoduchý i složený. Shell udržuje tabulku přezdívek a po spuštění příkazu nejdříve tuto tabulku prohlédne. Pokud zjistí, že spuštěný příkaz je přezdívka, nahradí jej příkazem, který přezdívce v tabulce přezdívek odpovídá.

Přezdívky byly původně zavedeny v C shellu. Původní Bourne shelly sice přezdívky neměly, ale ke stejnému účelu bylo možno použít uživatelsky definovaných funkcí. Současné verse Bourne shellu mají kromě funkcí i přezdívky. C shell možnost definice funkcí nemá.

Přezdívky i funkce lze definovat zadáním příkazu z příkazového řádku. Obvykle se ale příkazy, které je definují, vkládají do inicializačních souborů (**.cshrc** nebo **.profile**), aby byly po spuštění shellu okamžitě k dispozici.

Přezdívky v C shellu

Přezdívky se v C shellu definují příkazem **alias**

alias *název_přezdívky příkaz*

Výpis tabulky přezdívek lze získat příkazem **alias** , zadaným bez parametrů.

Příklad 13:

```
%alias h history
%alias  w 'date; who'
%alias
```

uvozovky jsou zde použity jako znaky výluky, aby shell neinterpretovat znak ;

```
h  history
w  date; who
```

Přezdívku lze napsat tak, aby ji bylo možno spouštět také s parametry. Například složený příkaz

ls -al [*directory ...*] **| more**

chceme nahradit přezdívkou

ll [*directory ...*]

V tomto případě v tabulce přezdívek musí být

```
ll   ls -al  !*|more
```

a přezdívku nutno zadat příkazem

```
%alias ll  'ls  -al  \!*|more'
```

Podobně lze zadat přezdívky pro zkrácený zápis hledání souboru **f** a výpis podaresářů **ldir** :

```
%alias f 'find / -name \!* -print'
%alias ldir 'ls -l \!* |grep "^d"|more'
```

Starší verze C shellu nejsou schopny po změně aktuálního adresáře měnit automaticky nastavení proměnné `prompt`, která obsahuje tvar výzvy. Tento jejich nedostatek lze odstranit pomocí aliasu následujícím způsobem:

Definuje se přezdívka **cd**, která kromě změny aktuálního adresáře navíc aktualizuje hodnotu proměnné `prompt`. Protože tabulka přezdívek se prohlíží dříve než se provádí

vnitřní příkazy shellu, provede se přezdívka **cd** a nikoliv vnitřní příkaz **cd**. Aby nedošlo k zacyklení při vyhodnocování přezdívky, použije se v definici přezdívky vnitřní příkaz shellu **chdir** , který je synonymem vnitřního příkazu **cd** . Alias lze tedy zadat takto:

```
%alias cd 'chdir \!* ; set prompt="[`hostname`:$user$cwd]"'
```

V proměnná `cwd` shell udržuje jméno aktuálního adresáře.

Přezdívky a funkce v Bourne shellu

Přezdívky se v Bourne shellu původně nepoužívaly. Moderní verse Bourne shellu jejich použití obvykle umožňují. Přesto se používají méně než v C shellu. Často se nahražují uživatelsky definovanými funkcemi, které plní stejné funkce, poskytují více možností a jejich realizace je jednodušší.

Přezdívku lze v Bourne shellu definovat příkazem **alias**

alias *název_přezdívky = příkaz*

Příklad 14:

```
$alias pp = 'ps -ael | more'
$alias
pp  ps -ael | more
```

Funkce se definují takto:

```
název_funkce()
{
  příkazy
}
```

Například v C shellu definovanou přezdívku **ll** (viz 6.8.1) lze v Bourne shellu nahradit takto:

```
$ll()
> { ls -al $* | more }
$ll

. . . . .
```

Definované funkce lze zobrazit příkazem **set .**

Definovanou funkci lze zrušit příkazem

unset *jméno_funkce*

Příklady

Příklad 15: Napište script, který smaže obrazovku, vypíše připojené uživatele a datum a čas. Výpis může mít přibližně tuto formu:

```
****************************************
Pripojeni uzivatele:
novak          tty1 Oct  12  16:40
novy           tty2 Oct  12  16:50
Datum a cas:
Mon Oct 12  17:10:20 GMT+0100 1998
****************************************
```

Řešení

```
#!/bin/bash
clear
echo "****************************************"
echo Pripojeni uzivatele
who
echo "Datum a cas: "
date
echo "****************************************"
```

Příklad 16: Jaký bude výsledek příkazů:

a)

```
%echo *
```

b)

```
%echo \*
```

(a) výpis aktuálního adresáře b) *)

Příklad 17: Zadali jste tuto sekvenci příkazů:

```
%hostname
beta
%setenv jmeno novak
```

Jaký dostanete výsledek, pokud potom zadáte příkaz:

a)

```
%echo "jmeno: $jmeno"
```

b)

```
%echo 'jmeno: $jmeno'
```

c)

```
%echo !ho
```

d)

`%echo `!ho``

(a) `jmeno: novak` b) `jmeno : $jmeno` c) `hostname` d) `beta`)

Příklad 18: Po příkazu

`%cat .cshrc .login`

zadáte příkaz

`%more !*`

Jak shell tento příkaz interpretuje?

(`more .cshrc .login`)

Příklad 19: Po příkazu

`%ls /etc | more`

zadáte příkaz

`%^/etc^/bin`

Jaký příkaz se provede?

(`ls /bin | more`)

Příklad 20: Po příkazu

`%ls -al`

zadáte příkaz

`%!! | more`

Jaký příkaz bude proveden?

(`ls -al | more`)

Příklad 21: V souboru `.cshrc` je uvedena tato řádka

`alias 'dir ls -al \!* | more '`

a) Jak bude shellem **csh** interpretován příkaz

`%dir /usr/bin /bin`

b) Lze v definici aliasu zaměnit jednoduché uvozovky za dvojité?

c) Lze v definici aliasu vynechat symbol \ ?

(a) `ls -al /usr/bin /bin | more` b) ano c) ne)

Příklad 22: V Bourne shellu nastavte posiční parametry na hodnoty :

```
$1   dnes
$2   je
$3   pekny
$4   den
```

Řešení

```
$ set dnes je pekny den
$ echo $3
pekny
$ echo $#
4
```

Příklad 23: V C shellu uložte do proměnné *a* seznam slov (dne je pekny den).

Řešení

```
%set a=(dnes je pekny den)
%echo $a[3]
pekny
%echo $#a
%4
```

Příklad 24: V C shellu vyhodnoťte následující výrazy :

1. $a = (x + y) * z$
2. $b = (x > y) \wedge (y > z)$

Řešení

1. `@ a=(($x + $y) * $z)`
2. `@ b=(($x > $y) && ($y > $z))`

Příklad 25: Výrazy z předcházejícího příkladu vyhodnoťte v Bourne shellu

Řešení

1.
```
a=`expr $x + $y`
a=`expr $a '*' $z`
```

2.
```
b1=`expr $x '>' $y`
b2=`expr $y '>' $z`
b=`expr $b1 '*' $b2`
```