evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

PSK3-2

Název školy:	Vyšší odborná škola a Střední průmyslová škola, Božetěchova 3
Autor:	Ing. Marek Nožka
Anotace:	Zpracování příkazového řádku unixovým shellem
Vzdělávací oblast:	Informační a komunikační technologie
Předmět:	Počítačové sítě a komunikační technika (PSK)
Tematická oblast:	Operační systém Linux/Unix
Výsledky vzdělávání:	Žák na příkladech ukazuje expanze metaznaků
Klíčová slova:	Linux, Unix, shell, expanze, jména souborů, alias, oddělování příkazů
Druh učebního materiálu:	Online vzdělávací materiál
Typ vzdělávání:	Střední vzdělávání, 4. ročník, technické lyceum
Ověřeno:	VOŠ a SPŠE Olomouc; Třída: 4L
Zdroj:	Vlastní poznámky, Vilém Vychodil: Linux Příručka českého uživatele

Zpracování příkazového řádku

Před spuštěním příkazu provádí shell nad příkazovou řádkou několik transformací:

Expanze proměnných

Proměnnou vytvoříme pomocí znaku =

```
PROMENA=data
```

Její obsah je možné vybrat pomocí znaku $:

```
ls $PROMENA
```

Existují i proměnné speciálního významu:

`$$` PID shellu

`$!` PID posledního procesu spuštěného na pozadí

`$?` návratová hodnota posledního dokončeného procesu

Expanze jmen souborů

Někdy je výhodné pro zápis jmen souborů použít žolíkové znaky (Wildcard character) Pod pojmem žolíkové znaky rozumíme následující konstrukce:

`\Z`	znak »Z«
`~`	domovský adresář
`~jmeno`	domovský adresář uživatele »jmeno«
`*`	libovolný počet libovolných znaků
`?`	právě jeden libovolný znak
`[abcd]`	znak `a` nebo `b` nebo `c` nebo `d`
`[a-d]`	znak `a` nebo `b` nebo `c` nebo `d`
`[a-de]`	znak `a` nebo `b` nebo `c` nebo `d` nebo `e`
`[a-bc-d]`	znak `a` nebo `b` nebo `c` nebo `d`
`[-ab]`	znak `a` nebo `b` nebo `-`
`[^a-ez]`	libovolný znak vyjma `a,b,c,d,e,z`
`[-a-c^z]`	jeden ze znaků `a,b,c,z,^,-`

Konstrukci v hranatých závorkách odpovídá vždy **jeden znak** z dané množiny. Znak `-` slouží pro zkrácený zápis množiny. Pokud je uveden na začátku `[-` nebo na konci `-]` odpovídá znaku `-`. Znak `^` uveden hned na začátku `[^` slouží jako negace. Pokud je uveden jinde odpovídá znaku `^`.

Je důležité zdůraznit, že **expanzi provádí shell nikoliv program**. Pokud napíše uživatel `mv * adresar`, je to shell a nikoliv program `mv`, kdo nahradí hvězdičku seznamem všech souborů v aktuálním adresáři.

Pokud chceme předat nějakému příkazu skutečně hvězdičku nebo jiný metaznak se zvláštním významem (např. mezera, která má zvláštní význam oddělování parametrů), je nutné předřadit mu zpětné lomítko `jmeno\ souboru` nebo dát metaznaky do uvozovek `"jmeno souboru"` nebo apostrofů `'jmeno souboru'`.

Shell při zpracování příkazové řádky apostrofy, uvozovky a zpětná lomítka odstraní, ale jejich obsah ponechá nezměněný. V apostrofech nemění vůbec nic, ale v uvozovkách stále nahrazuje proměnné a konstrukce, které začínají znakem `$`. *Přesné chování při zpracování příkazové řádky a seznam znaků se speciálním významem závisí na konkrétním shellu*.

Expanze speciálních konstrukcí

Na tomto místě rozhodně není uveden seznam všech konstrukcí, které lze použít. Vybírám jen ty nejpoužívanější.

Roznásobení

Do složených závorek můžeme zapsat čárkou oddělený seznam, kterým se roznásobí řetězec k němu přiléhající.

```
ls adresar{1,2,3}
```

Vypíše `adresar1 adresar2 adresar3`.

Vyzkoušejte si

```
echo {0,1}{0,1}{0,1}
echo 192.168.32.{1,2,3,4,5,6,7,8}
```

Výstup jiného příkazu

Někdy se hodí na příkazový řádek zadat výstup jiného příkazu. To je možné buď pomocí zpětných apostrofů `` `prikaz` `` nebo pomocí konstrukce `$(prikaz)`

Aktuální datum v zadaném formátu vypisuje program `date`. Například

```
date +%Y_%m_%d
    2013_10_17
```

Pokud bych požadoval vytvořit adresář, který se bude jmenovat podle aktuálního data použiji příkaz:

```
mkdir $(date +%Y_%m_%d)
```

Shell nejprve spustí příkaz `date` a jeho výstup potom umístí na příkazový řádek za `mkdir`.

Matematické výraz

Matematický výraz lze zapsat pomocí konstrukce `$[výraz]`.

```
N=123
N=$[ 2*$N ]
echo $N
echo $[ 2**10 ]
```

Návratová hodnota příkazu

Každý program, který na Unixu/Linuxu spustíme dává vědět o svém zdárném nebo nezdárném konci svou návratovou hodnotou. Pokud je návratová hodnota `0` program skončil úspěšně. Pokud je návratová hodnota od nuly různá skončil neúspěšně. Návratovou hodnotu naposledy vykonaného příkazu zjistíme přes proměnnou `$?`

```
ls data
echo $?
    0
ls nejakaBlbost
echo $?
    2
```

Nenechte se poplést tím, že úspěch je značen nulou. Čím větší je návratová hodnota, tím větší je neúspěch.

Program `true` má návratovou hodnoty vždy `0`. Program `false` má návratovou hodnotu vždy `1`.

Oddělování příkazů

Pokud chceme umístit na jeden řádek několik příkazů za sebe používá se jako oddělovač jednotlivých příkazů znak `;`.

```
sleep 5 ; ls data
```

Konstrukce `&&` a `||` slouží k podmíněnému vykonání příkazů.

```
mkdir texty && cp *.txt texty
```

Příkaz `mkdir` se vykoná vždy. Ale příkaz `cp` bude vykonán pouze pokud příkaz `mkdir` skončil úspěšně. Příkaz `&&` má význam logického součinu.

```
cd dddata || pwd
```

Příkaz `pwd` bude vykonám pokud příkaz `cd` skončí neúspěšně. `||` má význam logického součtu.

Vyhledání spustitelného souboru

Shell vyhledává program pro spuštění v cestách, které jsou v proměnné `$PATH`[1].

```
echo $PATH
    /usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Je-li zadán například příkaz `ls`, shell hledá postupně spustitelný soubor `ls` v adresáři `/usr/local/bin` potom v `/usr/bin` potom `/bin` atd. dokud takto pojmenovaný spustitelný soubor nenajde. Pokud ho najde, spustí ho. Pokud ho nenajde vypíše chybové hlášení.

Pokud chci například spustit svůj vlastní program uložený v aktuálním adresáři

```
mujprogram.py
```

```
bash: mujprogram.py: příkaz nenalezen
```

shell program nespustí, protože aktuální pracovní adresář není v proměnné `$PATH`. Je proto nutné shellu říct, kde program je. To udělám tak, že k němu napíšu plnou cestu. Například:

```
./mujprogram.py
```

nebo

```
/home/uzivatel/bin/mujprogram.py
```

Pokud chceme vyhledat, kde se program nachází slouží k tomu příkaz `which` nebo `ls`.

```
type -a ls
    ls je alias na „ls -F --color=auto“
    ls je /bin/ls
```

Zjistíme, že spuštěním příkazu ls se nespouští `/bin/ls` jak by se podle dříve řečeného zdálo ale tzv. **alias**. Rozdíl můžete porovnat pokud zadáte „holý“ příkaz `ls` a když zadáte plnou cestu `/bin/ls`.

Aliasy

Alias[2] definuje zkratku pro jiný (dlouhý) příkaz. Typicky se vytváří aliasy:

```
alias ls='ls -F --color=auto'
alias ll='ls -l'
```

Alias „žije“ pouze po dobu běhu příkazového interpretu a po jeho ukončení je zapomenut. Proto se aliasy ukládají do souboru, který se automaticky spouští při startu shellu. Pro Bash je to soubor `~/.bashrc`.

---

1. ~/.bashrc ↩
2. ~/.bashrc ↩