

# PV004 UNIX

Michal Brandejs  
brandejs@fi.muni.cz

Katedra počítačových systémů a komunikací FI MU  
Centrum výpočetní techniky FI MU

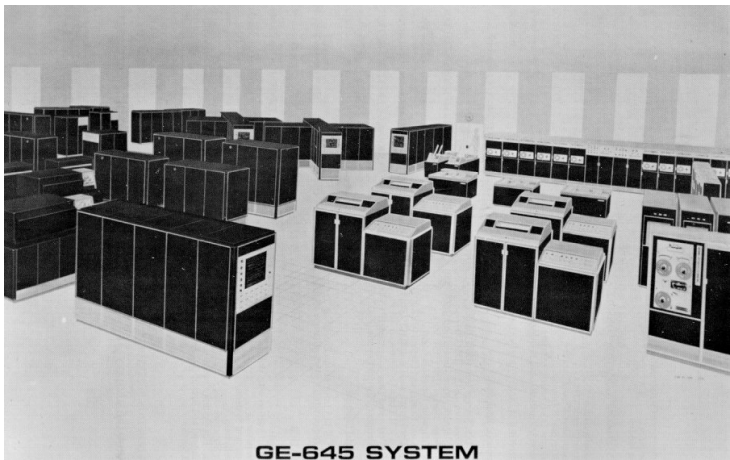
17. února 2021

# Schéma přístupu k UNIXu

**Terminály** v kancelářích připojené sériovou RS-232 linkou k **serveru** uzavřeném na počítačovém sále.



# Pre-UNIX Time



1965-69

Pokus o sw projekt **Multics** (Bell Labs, GE, MIT).

# Pojem "konzole"

Terminál připojený přímo k počítači. Příklad: IBM 100 Systém 360



# Historie UNIXu

Minipočítače – PDP-7, údajně použité pro první unixový kód



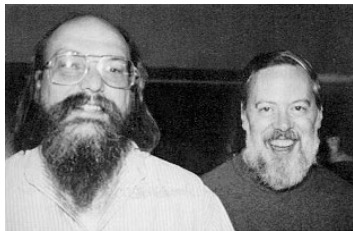
Vnitřní paměť 4 Kslov (18bitové slovo, 9 KB) až 32 Kslov, disk 1 Mslov

# Historie UNIXu

1970

**Ken Thompson** na papíře vytváří nový mechanismus multiuživatelské správy souborů.

**Denis Ritchie** a K.T. jej implementují na počítači PDP-7.



Ken Thompson a Denis Ritchie

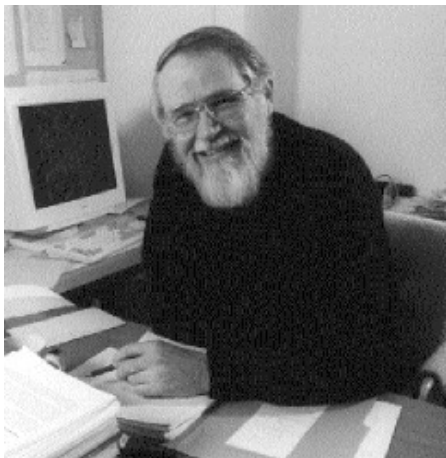
1971

Přesvědčí Bell Labs o vhodnosti zakoupit PDP-11/20 pro vývoj systému na zpracování textů pro patentové oddělení. (Ken Thompson sedí.)



1971

Vzniká název **UNIX** (údajný autor **Brian Kernighan**)





1971

K. Thompson zahajuje implementaci Fortranu pro UNIX; vzniká jazyk **B**.

D. Ritchie přidává datové typy se strukturami a vzniká jazyk **C**.

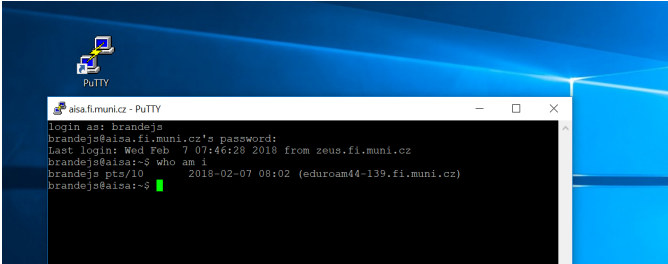
1973

Většina operačního systému přepsána do jazyka C.

# Jak použít studentský server Aisa?

- Např. programu PuTTY
- vybrat protokol **ssh**
- spojit se na počítač **aisa.fi.muni.cz**
- login a heslo:

<https://is.muni.cz/auth/edutools/brandejs/pv004lab> ▶ Link



The image shows a Windows desktop with a blue background. A PuTTY window is open, titled "aisa.fi.muni.cz - PuTTY". The terminal output is as follows:

```
login as: brandejs
brandejs@aisa.fi.muni.cz's password:
Last login: Wed Feb  7 07:46:28 2018 from zeus.fi.muni.cz
brandejs@aisa:~$ who am i
brandejs pts/10      2018-02-07 08:02 (eduroam44-139.fi.muni.cz)
brandejs@aisa:~$
```

# Kde je Aisa?

V nejvyšším patře budovy A

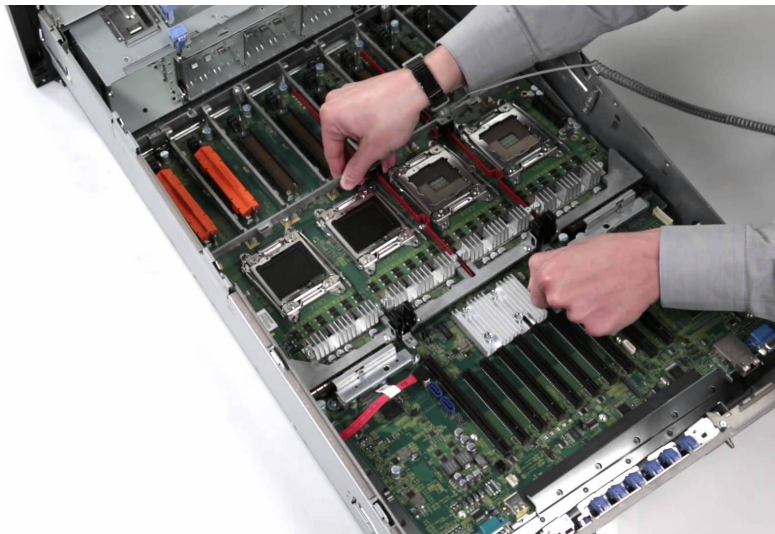


# Co je Aisa?

od února 2017: Dell PowerEdge R930  
4x šestnáctijádrový procesor Intel® Xeon® E7-8860 v3  
0,5 TB operační paměť; 4x 480 GB SSD disk



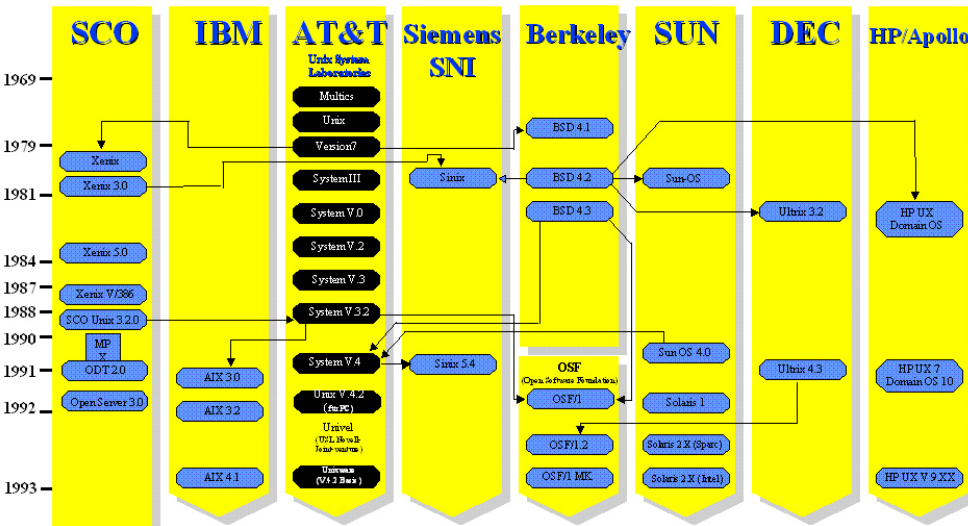
# Aisa: Dell PowerEdge R930



# Ochranná známka UNIX

- Bell Labs
- AT&T
- USL UNIX System Laboratories (1991)
- Novell
- X/Open (1993)
- Open Group (součástí je X/Open) (1996)
- 1995 Novell → Santa Cruz Operation (sporné)  
Santa Cruz Operation → Caldera  
Caldera se přejmenovala na SCO The Open Group

"UNIX is a registered trademark of The Open Group"



Převzato z The Open Group, *The UNIX System – What about all those "Flavors"?* [online], c1995-2007,

Disponibilní z WWW: <[http://www.unix.org/what\\_is\\_unix/flavors\\_of\\_unix.html](http://www.unix.org/what_is_unix/flavors_of_unix.html)>

# Pravidla vývoje UNIXu

Vývoj UNIXu se od počátku řídil těmito pravidly:

- Psát programy, které budou dělat právě jednu věc, a tu budou dělat dobře.
- Psát programy tak, aby mohly navzájem spolupracovat.
- Psát programy tak, aby povelů přijímaly hromadně ze vstupu v textové podobě.
- Psát programy tak, aby výstupy produkovaly v textové podobě a mohly být použity jako vstupy do programů dalších.



# Příčiny popularity UNIXu

- Systém je napsán programovacím jazykem vyšší úrovně.
- Jsou dostupné zdrojové texty systému.
- Má jednoduché a zdokumentované uživatelské rozhraní.
- Nabízí prostředky na budování komplexních programů z jednodušších.
- Poskytuje jednoduché konzistentní rozhraní periferních zařízení.
- atp.

# Vlastnosti operačního systému UNIX

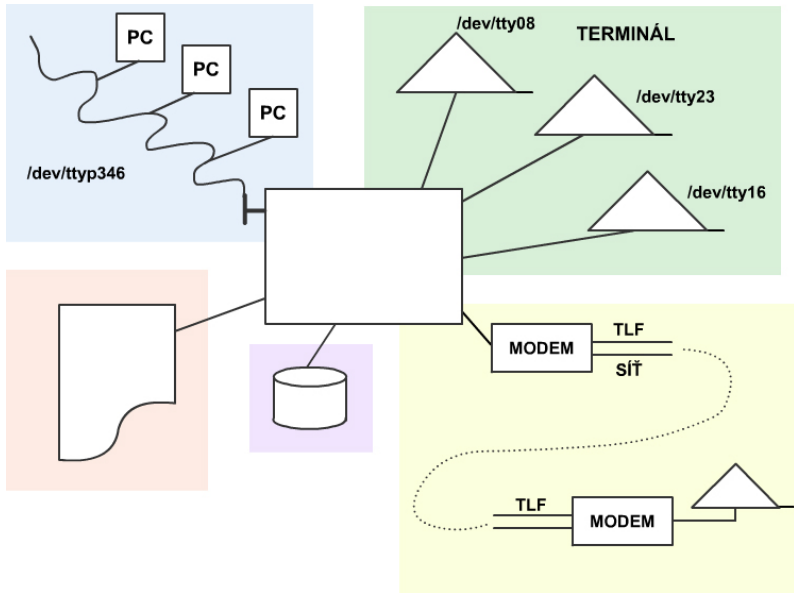
- multiprogramový
- multiuživatelský
- s terminálovým přístupem

Operační systém musí uživateli u terminálu vytvořit pocit, že počítač schovaný za terminálem je "pouze" jeho.

- ochrana uživatelského prostředí
- ochrana souborů
- ochrana procesů
- ochrana systému proti uživatelům

Pojem:

- démon



# Prvky ochrany uživatelů v multiuživatelském OS UNIX

## Účet uživatele:

- přihlášení
- práce v sezení (relaci)
- odhlášení

## Účet obsahuje: `/etc/passwd`

### Uživatelské jméno

login name, 3-8 znaků, začíná  
písmenem, jedinečné v systému, záleží  
na velikosti písmen

### Heslo

password

### Uživatelské číslo

user identification UID

## Účet obsahuje: (pokračování)

### Primární skupina

group GID

### Doplňující identifikace uživatele

jméno, příjmení, ...

### Domácí adresář

### Shell (interpret příkazů)

/bin/sh, /bin/ksh, /bin/bash, ...

### Další informace

čas posledního přihlášení, doba platnosti  
hesla, ... (záleží na implemtaci)

## Práva pro přístup k souborům a adresářům:

určují se zvlášť pro

majitele souboru (u - user)

skupinu uživatelů (g - group)

ostatní, svět (o - other)

zvlášť pro

čtení/zápis/provedení (soubor)

výpis/modifikace/vstup (adresář)

## Speciální uživatel root (superuživatel, UID=0)

Jediný uživatel v systému, kterému se přístupová práva nekontrolují.

# Přístup uživatele k systému: přihlášení

login: xnovak

uživatelské jméno, malými písmeny; nemusí být znám typ terminálu - proto nemusí fungovat znaky BACKSPACE apod.

password: moje heslo

neopisuje se, libovolné znaky vč. mezer a CTRL znaků; musí být netriviální; alespoň 6 znaků (dříve bylo významných jen 8 prvních). Heslo nelze zpětně zjistit.

Při chybném přihlášení je vncucena prodleva a vypíše se:

Login incorrect

login:



## Při úspěšném přihlášení:

vypíše se systémové zprávy

vypíše se případné sdělení správce systému

domovský adresář se nastaví jako běžný (pracovní)

spustí se požadovaný shell

vypíše se výzva shellu (prompt)

\$

Prompt závisí na nastavení shellu (proměnná PS1)

`set | grep PS1`

`exit` nebo `^D` (může být potlačeno)

# Zadávání příkazů shellu

- Dělají se rozdíly mezi malými a velkými písmeny
- Shell interpretuje příkazy po stisku Enter
- Příkazy lze zadávat "do zásoby"
- Některé shelly si pamatují historii příkazů s možností editace a opětovného použití
- Dokud řádek není ukončen stiskem Enter, lze
  - smazat poslední znak stiskem znaku **erase**
  - zrušit celý řádek stiskem znaku **kill**
- Popis vlastností terminálů: `/etc/terminfo/*/*`, příp. `/lib/terminfo/*/*`, dříve soubor `/etc/termcap`

Nastavení speciálních znaků zjistíme příkazem:

```
stty -a
```

Nezobrazitelné znaky se v UNIXu vypisují:

- jako `^C` ... ctrl-C (ordinální hodnota 3)
- nebo `\012` ... tři číslice oktalově

```
bash$ stty -a
intr = ^C; quit = ^ \; erase = ^?; kill = ^U;
eof = ^D; eol = <undef>; eol2 = <undef>;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R;
werase = ^W; lnext = ^V; flush = ^O; min = 1;
time = 0;
```

**intr**

Násilně ukončí běžící proces ( ^C )

**eof**

Ukončení vstupu (např. z klávesnice; ^D; End of File; EOF)

**stop**

Pozastavení výpisu (např. na obrazovku; ^S; X-off)

## start

Pokračování ve výpisu (^Q; X-on)

## susp

Pozastavení procesu (resp. jeho převedení pod job control; ^Z). Používejte uvážlivě - je nebezpečné zapomínat procesy.

Příklad:

```
ping aisa
```

```
^Z
```

```
ps
```

```
kill -9 11111
```

```
# 11111 je číslo procesu ping z výpisu ps
```

- počítá se v sekundách
- uloženo na 32 bitech se znaménkem
- "epocha" od 1. 1. 1970
- konec 32bitového světa: 03:14:08 UTC on 19 January 2038

# Nápověda - man (on-line manual pages)

Nápověda k určitému příkazu:

```
man passwd
```

```
man stty
```

```
man man
```

Reference ve tvaru: man(1)

jméno příkazu(sekce manuálu UNIXu)



## Sekce

- 1 Příkazy uživatelské úrovně
- 2 Systémová volání
- 3 Knihovní funkce
- 4 Zařídění a ovladače zařízení
- 5 Formáty (konfiguračních) souborů
- 6 Hry
- 7 Různé (ASCII), popisy maker
- 8 Nástroje pro údržbu systému

Sekce se zadává:

`man passwd` (nebo `man 1 passwd`)

vypíše návod k příkazu **passwd(1)**  
pro změnu hesla

`man 5 passwd`

vypíše návod k formátu souboru  
`/etc/passwd` **passwd(5)**

Zadávání sekce v jiných implementacích:

**man -s 3 tzset**

**man -S 3 tzset**

# Systémy souborů

Různé systémy souborů:

Základní pro UNIX:

ufs (Solaris), efs (IRIX), ext2fs (Linux)  
a další

Doplňující:

msdos, vfat, nfs (Network fs), iso9660,  
swap, tmpfs a další

Vyrovňávání diskových operací:

"Špinavé" bloky zapisuje na disk každých 30 vteřin  
démon **bdflush**, **update**, **fsflush** apod. jmen.

## Systém souborů:

- Zaváděcí blok
- Superblok
- Seznam i-uzlů
- Seznam datových bloků

# Jméno souboru a adresáře

- Jméno souboru smí být dlouhé nejvýše **255 znaků**. Na tuto délku je zkracováno bez varování.
- Může obsahovat libovolné znaky. Komplikace způsobují řídicí znaky shellu. Výjimka: v kořenovém adresáři nesmí být soubor jména `"/`.
- Rozlišují se malá a velká písmena.
- Zvláštní význam mají jména souborů začínající znakem `"."`(tečka), např. `".profile"`. Při expanzi nahrazovacího znaku `"*"` se taková jména nepoužijí. Vyzkoušejte též `"ls"` a `"ls -a"`.

# Adresáře

- Oddělovačem jmen adresářů je znak "/" (lomítko).
- **Kořenový adresář** (root) "/"
- **Běžný** (pracovní) **adresář**
- Rozlišujeme cestu **absolutní** "/.../.../..." a **relativní** ".../.../...".
- Každý adresář obsahuje položky **"."** (**tento adresář**) a **".."** (**nadřazený adresář**)

# Příklady

`pwd`

Print Working Directory (zjištění cesty k běžnému adresáři)

`cd adresář`

Change Current Directory (bez parametru nastaví domovský)

**`cd ..`**

**`./program`**

`mkdir adresář ...`

Make Directories

`rmdir adresář ...`

Remove Directories (lze rušit pouze prázdné adresáře)

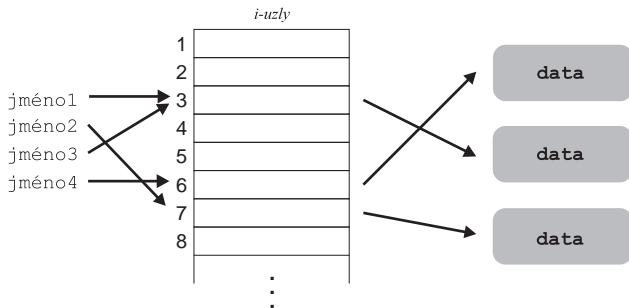
# Repräsentace souboru na disku

## Obsah souboru:

Z pohledu UNIXu je soubor posloupností bajtů. Systém obsah souboru neinterpretuje. Poznámka: V textovém souboru se řádky oddělují znakem LF (vs. MS-\* odděluje řádky znaky CR LF).

- Informace o souborech jsou soustředěny do jednoho místa na paměťovém médiu – do **seznamu i-uzlů** (i-node).
- Jeden soubor (nebo adresář) je popsán právě jedním i-uzlem.





- *i-uzly* neobsahují jméno souboru.
- "**ls -il**" vypíše obsah adresáře vč. čísla *i-uzlu*.
- ***i-uzel* číslo 2** ukazuje vždy na kořenový adresář.

# Formát adresáře

## Kořenový adresář a jeho podadresář:

<i>jméno</i>	<i>i-uzel</i>
.	2
..	2
jméno1	3
jméno2	7
jméno3	3
jméno4	6
prog	87
	0
⋮	

<i>jméno</i>	<i>i-uzel</i>
.	87
..	2
prog.c	93
a.out	98
	0
	0
	0
	0
⋮	

Prázdná položka má číslo i-uzlu nulové.

## Příkazy:

### Vytvoření souboru:

Mnoho způsobů; např. "**touch soubor ...**"  
(**prázdný soubor se vytvoří pouze vedlejším efektem tohoto příkazu**)

### Zrušení souboru:

**rm soubor ...**

### Kopie souboru:

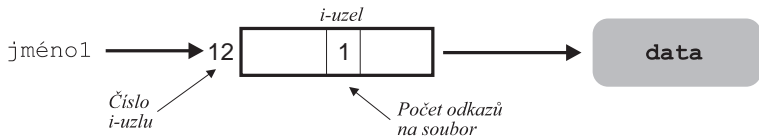
**cp zdroj cíl**

### Přesun/přejmenování souboru/adresáře:

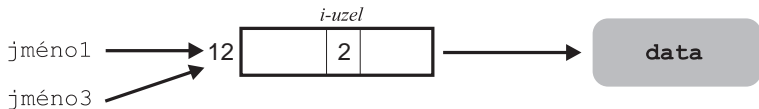
**mv zdroj cíl**

# Odkaz (Tvrký odkaz)

Situace: vytvořili jsme soubor "jméno1"



**In jméno1 jméno3**

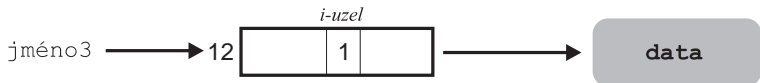


Zjištění počtu odkazů na souboru: **ls -l**

```
-rw-r--r-- 2 brandejs users 5 Mar 10 21:25 jméno1  
-rw-r--r-- 2 brandejs users 5 Mar 10 21:25 jméno3
```

- Hodnota "2" je počet odkazů na soubor.
- Úvodní "-" (minus) znamená, že jde o obyčejný soubor.
- Položky "jméno1" a "jméno3" jsou rovnocenné.

## rm jméno1

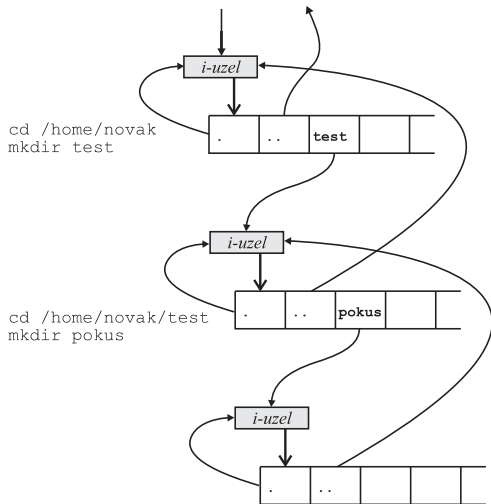


## ls -l

```
-rw-r--r-- 1 brandejs users 5 Mar 10 21:25 jméno3
```

- Nelze vytvářet (tvrdé) odkazy na adresáře. Proč?
- Odkazy lze dělat pouze uvnitř systému souborů. Proč?
- Komu patří vlastnictví nově vytvořeného odkazu?

# Odkazy a adresáře



```
bash$ ls -ld test
drwxr-xr-x 3 brandejs users 1024 Mar 10 22:09 test
bash$ cd test
bash$ ls -ld pokus
drwxr-xr-x 2 brandejs users 1024 Mar 10 22:09 pokus
bash$
```

- Úvodní "d" znamená, že jde o adresář.
- V případě nejasností lze použít "**ls -ild**".

```
bash$ ls -ild pokus
81937 drwxr-xr-x 2 brandejs users 1024 Mar 10 22:09 pokus bash$ cd
pokus
bash$ ls -ild .
81937 drwxr-xr-x 2 brandejs users 1024 Mar 10 22:09 .
bash$
```

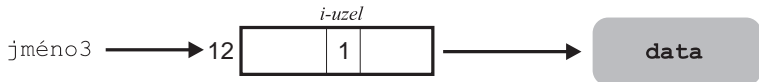


# Symbolické odkazy

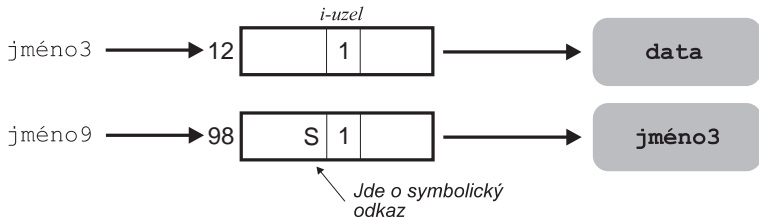
## **Symbolické odkazy umožňují:**

- Odkazy na adresáře
- Odkazy mimo jeden systém souborů

Situace: máme soubor **jméno3**



In -s jméno3 jméno9

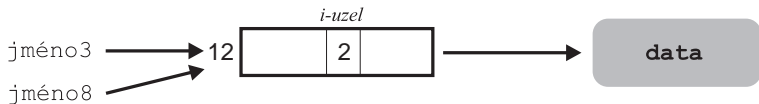


## ls -l

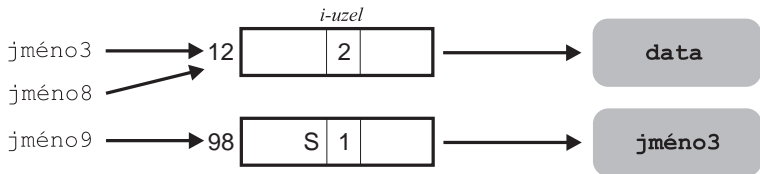
```
-rw-r--r-- 1 brandejs users 5 Mar 10 22:46 jméno3  
lrwxrwxrwx 1 brandejs users 6 Mar 10 22:48 jméno9 -> jméno3
```

- Úvodní "l" (písmeno L) znamená, že jde o symbolický odkaz.
- Hodnota 6 (příp. 7 při použití utf-8) je velikost souboru, tj. počet bajtů "jméno3".
- Položky "jméno3" a "jméno9" nejsou v žádném vztahu.

Výjimka: Zrušení symbolického odkazu **rm jméno9**

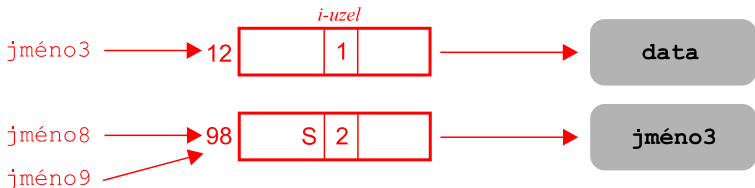


## In jméno9 jméno8



Jde o textovou substituci, kterou se výskyt "jméno9" nahradí "jméno3". **V aktuálních verzích jádra změněno na:**

## In jméno9 jméno8



i-uzel (i-node, "i" jako index)

- i-uzel na disku
- i-uzel v paměti

na disku

- volný x obsazený
- ID vlastníka souboru
- ID skupiny vlastníků
- typ souboru (obyčejný, adresář, znakový nebo blokový speciální soubor, FIFO)
- přístupová práva

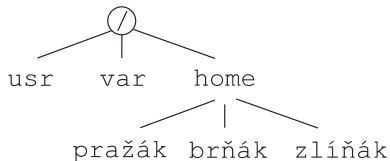
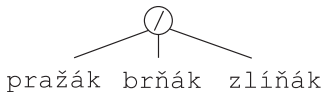
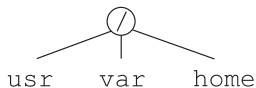
## na disku (pokračování)

- datum a čas poslední modifikace obsahu souboru, modifikace i-uzlu, přístupu k souboru
- počet odkazů na soubor
- seznam diskových adres uložení souboru
- velikost souboru v bajtech

## paměťová kopie obsahuje navíc

- identifikaci uzamknutí (pro výlučný přístup)
- ...
- počet odvolávek (např. kolik instancí souboru je otevřených)
- soubor je místem připojení dalšího systému souborů





Na disku jsou i-uzly v jednorozměrném poli  $\langle 1 .. \max \rangle$ , v paměti jsou zřetězeny do seznamu.

# Superblok

- velikost systému souborů
- počet volných bloků v systému souborů
- seznam volných bloků
- index následujícího volného bloku v seznamu
- velikost seznamu i-uzlů
- počet volných i-uzlů
- seznam volných i-uzlů
- index následujícího volného i-uzlu v seznamu
- pole zámků pro seznam volných bloků a i-uzlů
- příznak modifikace superbloku

# Speciální soubory

- Rozlišujeme:
  - Znakový speciální soubor
  - Blokový speciální soubor
- Součástí jádra jsou "ovladače zařízení"
- Uživatelské rozhraní vůči zařízením se realizuje prostřednictvím systému souborů
- **Speciální soubor** je odkaz na i-uzel, který ukazuje na ovladač

Příkaz pro vytvoření speciálního souboru:

```
mknod jméno typ hlavní_číslo vedlejší_číslo
```

typ

je "b" (blokové zařízení) nebo "c" (znakové zařízení)

hlavní\_číslo

je číslo udávající typ zařízení – ukazuje do tabulky zařízení

vedlejší\_číslo

je číslo jednotky

- Znakový a blokový speciální soubor vytváří superuživatel
- Zpravidla uloženy v adresáři **/dev**
- Ve výpise příkazu **ls -l** se objeví (údaje v pořadí hlavní číslo, vedlejší číslo) např.:

```
crw-rw-rw- 1 root sys 14, 4 Apr 25 1995 audio  
brw-rw-r-- 1 root mail 2, 0 Jan 1 1980 fd0
```

# Příklady speciálních souborů

## Příklady z Linuxu:

`/dev/console`

konzola počítače (obrazovka a klávesnice),

`/dev/fd0`

disketa číslo 0,

`/dev/hda1`

první IDE disk a jeho první oblast,

`/dev/sda1`

první SCSI disk a jeho první oblast,

`/dev/lp1`

paralelní rozhraní Centronics odpovídající LPT1:,

## Další příklady z Linuxu:

`/dev/cua0`

sériové rozhraní RS-232 odpovídající COM1.,

`/dev/tty1`

první virtuální konzola,

`/dev/null`

prázdné zařízení, které přijme všechna data a které při čtení předá ihned **eof** (konec souboru).

## Příklady ze Systému V:

`/dev/dsk/c0t2d0s6`

SCSI zařízení – disk z čísla řadiče 0, SCSI adresy 2, SCSI LUN 0 a oblasti 6

`/dev/pty/78`

pseudo terminal driver (pty, pts)

`/dev/rst12`

magnetická páska (před zápisem souboru se vždy převine na začátek)

`/dev/nrst12`



## Speciální soubory (2)

FIFO – pojmenovaná roura

Vytváří se příkazem:

```
mknod jméno p
```

- p (pipe) roura
- Pro přenos dat mezi procesy
- Při zápisu do roury se data ukládají na disk téměř stejně jako při zápisu do souboru
- i-uzel obsahuje ukazatel pro čtení a ukazatel pro zápis
- Ukazatele nelze měnit jinak než čtením/zápisem – důsledně FIFO

# Příklad:

mknod trubka p

ls -l

```
prw-r--r-- 1 brandejs users 0 Mar 18 22:12 trubka
```

man mknod > trubka &

Přesměrováním > výstup nepůjde na obrazovku, ale do souboru "trubka". Ukončením příkazu znakem & jej spustíme na pozadí.

more trubka

Výpis obsahu souboru po obrazovkách.

rm trubka

Zrušení položky "trubka" v adresáři a zrušení i-uzlu.

# Typický adresářový strom

`/unix, /bsd, /boot`

Jádro OS

`/bin/ls, cp, sh`

Základní systémové programy a příkazy

`/dev`

Adresář speciálních souborů

`/etc`

Adresář většinou konfiguračních souborů systému

`/lib`

Adresář knihoven

`/mnt`

Pomocný adresář pro připojování dočasných systémů souborů

/tmp

Veřejný adresář pro pomocné a dočasné soubory

/home

Adresář s domovskými adresáři uživatelů

/usr/bin, etc, lib, tmp

Adresáře se soubory, které typicky z kapacitních důvodů nejsou v kořenovém adresáři.

/usr/include

.h soubory pro překladač jazyka C

/usr/man

Manuálové stránky

/usr/local/bin, man, etc, lib, ...

Programy lokálně instalované

## Typický adresářový strom (2)

`/usr/sbin`

Systemové programy určené zpravidla superuživateli

`/usr/X11, openwin`

Okénkový systém

`/var`

Adresář pracovních/administrativních/... souborů  
systému

`/var/mail`

Poštovní schránky uživatelů

`/var/spool`

Dočasné soubory systémových operací

`/var/adm`

Záznamy o činnosti systému a uživatelů

`/var/tmp`

`/var/preserve`

Pracovní soubory editoru **vi**, které zůstávají při násilném ukončení editoru.

# Přístupová práva

V i-uzlu pro vyhodnocení přístupových práv jsou:

## **vlastník souboru**

Číselné UID toho uživatele, který soubor vytvořil nebo kterému jej superuživatel věnoval.

## **skupina vlastníka**

Číselné GID skupiny, do které byl uživatel v okamžiku vytváření souboru přihlášen nebo na kterou bylo GID změněno.

## **přístupová práva**

Přístupová práva jsou v objektu uložena ve 12 bitech.

Ve výpisu **ls -l**

- 

rwx	r-x	r-x
-----	-----	-----

**1 novák student ...**

**vlastník** (označuje se u jako 'user')

Vlastníkem je ten, jehož UID je zapsáno v i-uzlu.

**skupina** (označuje se g jako 'group')

**ostatní** (označuje se o jako 'others')

Touto kategorií určujeme, co s tímto objektem mohou provádět všichni.

Přístupová práva se vyhodnocují v pořadí ugo.



Pro **soubor** se operace definují následovně:

- r Soubor je povoleno číst.
- w Do souboru je povoleno zapisovat.
- x Soubor je povoleno spustit (provést).

## Pro adresáře mají operace tyto významy:

- r** Adresář je povoleno vypsát; nikoli však zpřístupnit soubory v něm odkazované.
- w** Do adresáře je povoleno zapisovat; tj. lze vytvářet a rušit soubory.
- x** Do adresáře je možné vstoupit; tj. adresář může být argumentem příkazu **cd** a lze zpřístupnit i-uzly souborů, na které se adresář odkazuje.

## Příklady vyhodnocení přístupových práv:

- Vytvořit soubor mohu v kterémkoli adresáři, do kterého mám právo vstupu (x) a současně zápisu (w).
- Zrušit soubor libovolného vlastníka mohu v libovolném adresáři, do kterého mám právo vstupu (x) a současně zápisu (w).

Toto je **problém** zvláště u adresářů typu **/tmp**

```
drwxrwxrwx 9 root root 2048 Mar 18 22:12 tmp
```

Pro tyto adresáře se zavádí tzv. **sticky bit**:

```
drwxrwxrwt 9 root root 2048 Mar 18 22:12 tmp
```

Soubor v takto označeném adresáři může zrušit pouze vlastník souboru nebo vlastník adresáře.

- Číst soubor mohu, pokud mám právo vstupu do adresáře (x) a právo čtení souboru (r).
- Zapisovat do souboru mohu, pokud mám právo vstupu do adresáře a právo zápisu do souboru (w).
- Spustit soubor mohu, pokud mám právo vstupu do adresáře a právo provedení souboru (x). **Soubor běží pod UID toho, kdo jej spustil.** Jiné možnosti:
  - Chci, aby soubor běžel pod UID vlastníka souboru bez ohledu na to, kdo jej spustil: použiji **Set User ID (SUID)** bit **rwsr-xr-x**
  - Totéž pro skupinu: použiji **Set Group ID (SGID)** bit **rwxr-sr-x**

- **Přístupová práva jsou uložena ve 12 bitech**

### Osmičkový popis práv:

4000	SUID
2000	SGID
1000	sticky bit
0400	r pro vlastníka
0200	w pro vlastníka
0100	x pro vlastníka
0070	rwX pro skupinu vlastníků
0007	rwX pro ostatní

Všechny kombinace jsou možné, ne však významné.

Příklady symbolického zápisu příst. práv:

`rwsr-sr-x (6755)`

`rwxrwxrwt (1777)`

`rwxrwxrwt (1776)`

`rwSr--r-- (4644)`

# Příkaz umask

- Zadávání implicitních přístupových práv pro vytváření souborů a adresářů.
- Interní příkaz shellu.
- **umask nnn**
- Zadává se zpravidla numericky třemi osmičkovými číslicemi (nejvýše třemi)
- Masky se zadávají inverzně
- **umask 022**  
touch soubor (rw-r--r--)  
mkdir adresář (rwxr-xr-x)
- **umask 027**  
touch soubor (rw-r-----)  
mkdir adresář (rwxr-x---



# Příkaz chmod

chmod mode soubor ...

Nastavení přístupových práv.

Kdo smí přístupová práva nastavovat?

mode

- absolutně vyjádřeno 1-4 osmičkovými číslicemi
- symbolicky - lze popsat následující gramatikou:

```
mode ::= clause[,clause ...]  
clause ::= [who ...][action ...]last_action  
action ::= op[perm ...]  
last_action ::= op[perm ...]  
who ::= a|u|g|o  
op ::= +|-|=  
perm ::= r|s|t|w|x|X|u|g|o
```

# Příkaz chmod

who:

'u' (user), 'g' (group), 'o' (other); 'a' (all) je totéž co 'ugo'. Je-li who nezadáno, potom je to totéž co 'a' (all), ale nenastaví se bity označené příkazem **umask**.

perm:

'X' je totéž co 'x', ale aplikuje se pouze na adresáře nebo na soubory, které mají alespoň jeden bit 'x' nastavený.

'u', 'g', 'o' použije se současné nastavení příst. práv uživatele, skupiny, ostatních.

op:

'+' přidávají se práva, '-' ubírají se práva, '=' absolutní nastavení.

# Příkazy pro nastavování příst. práv

## Příklady použití příkazu `chmod`:

`chmod go-w soubor`

zakáže zápis pro skupinu a ostatní

`644`

nastaví `rw-r--r--`

`=rw,+X`

nastaví 'rw' podle **umask** a 'x' pro všechny ('ugo')  
tehdy, je-li soubor proveditelný alespoň pro jednoho  
z 'ugo' nebo jde o adresář.

## Příklady použití příkazu chmod:

`go=`

smaže všechny bity pro 'go'

`a+rwx`

typické nastavení pro veřejný pracovní adresář

`u+s`

nastavení SUID

`g+s`

nastavení SGID (majitel souboru musí být členem skupiny)

# Příklady

```
chmod_g+w_jedna_dva
```

```
go-rwx,u+x
```

```
ugo=rwxt
```

```
u+s
```

```
g+s
```

```
chmod_g+rx_*
```

```
g+X *
```

## chown vlastník soubor ...

Změna vlastníka souboru nebo adresáře.

Vlastník se zadává buď už. jménem, nebo číselně  
UID.

Jak zjistím svoje UID?

Vlastníka souboru smí měnit pouze superuživatel.

POSIX 1003.2 povoluje syntaxi:

*chown vlastník:skupina soubor ...*

pro současnou změnu vlastníka i skupiny.

## newgrp skupina

Přihlášení do jiné skupiny.

Příkazem se mění aktuální GID přihlášeného uživatele.

Uživateli je přihlášením do systému nastaveno jisté GID (zaznamenáno v účtě) a tímto příkazem může uživatel svoje GID měnit na jedno z těch, které má povoleno v /etc/group.

Příkaz bez uvedení skupiny nastaví GID podle účtu.



## chgrp skupina soubor ...

Změna skupiny souboru nebo adresáře.

Skupinu smí měnit vlastník souboru nebo superuživatel. Vlastník souboru však pouze tehdy, pokud náleží do cílové skupiny.

# Uživatelské rozhraní

Známé:

- Bourne shell (**sh**)
- C-shell (**csh**)
- Korn shell (**ksh**)
- Bourne-again shell (**bash**)

Co je shell?

Shell je interpretační programovací jazyk. Čte příkazy z terminálu nebo ze souboru a provádí je.

## Obecný formát příkazového řádku:

arg0 arg1 arg2 ...

Řetězec arg0 je vždy jméno příkazu.

### Příkazy:

- vnitřní
- vnější

arg0 se nejprve hledá v seznamu vnitřních příkazů a potom podle seznamu cest proměnné PATH

### Ukončovací kód (exit status):

Každý příkaz při svém ukončení vrací numerický stav. Hodnota 0 znamená OK. Nenulová hodnota značí chybu.

## Lexikální analýza příkazového řádku

Shell čte ze vstupu "slova" vzájemně oddělená "bílým místem" (mezera, tabulátor) a operátory:

Řídící operátory:

& && ( ) ; ;; | || nový řádek

Operátory přesměrování:

< > >| << >> <& >& <<- <>

# Řídicí znaky

## Obrácené lomítko (backslash):

Znak následující za \ ztrácí svůj řídicí charakter a stává se normálním znakem.

Výjimka: \ na konci řádku (\nový řádek) sděluje, že další řádek je pokračováním tohoto.

## Apostrof (single quote):

Uzavřením znaků mezi dva apostrofy '...' znaky ztrácejí svůj řídicí význam (vyjma ').

## Úvozovky (double quotes):

Uzavřením znaků do dvojice úvozovek "..."ztrácejí znaky svůj řídicí význam vyjma:

- \$ (dolar)
- ' (obrácený apostrof)
- \ (obrácené lomítko)
- Obrácené lomítko uvnitř dvojice úvozovek lze použít pouze pro:

```
\$ \` \" \\\ \nový řádek
```

Jinak se \`\` interpretuje jako normální znak.

Vyzkoušejte si příkazem echo

```
echo \'
echo '\\
echo \\
echo \"
echo \"\'
touch muj\ soubor
```

# Přesměrování vstupu a výstupu

Shell přesměrování–propojení zajistí před provedením příkazu.

File descriptor number (n)

příkaz n> soubor

příkaz n< soubor

Každý proces má nastaveno:

0 tzv. standardní vstup

1 tzv. standardní výstup

2 tzv. standardní chybový výstup

Další si nastavuje programátor.

Implicitní varianty:

1> je totéž co >

0< je totéž co <



## Přesměrování vstupu:

příkaz `n < soubor`

- Soubor musí existovat

## Přesměrování výstupu:

příkaz `n > soubor`

- Pokud soubor neexistuje, vytvoří se
- Pokud soubor existuje, v okamžiku otevření se zkrátí na velikost 0 a bude se přepisovat
- Pokud však byl proveden příkaz `"set -C"` (nebo `"export noclobber="`), potom nelze přesměrovat výstup do existujícího souboru. Toto omezení odstraní operátor `">|"`.

# Přesměrování vstupu a výstupu

Příklady: `$ ls neni je`

```
ls: neni: No such file or directory
je
```

```
$ ls neni je > a 2> b
```

**Nesmí se použít:** `ls > a > b`

Soubor 'a' zůstane prázdný, soubor 'b' se naplní stand. výstupem. Problém lze řešit např.:

```
ls|tee a > b
```

Připojení výstupu na konec souboru

příkaz `n>> soubor`

- Pokud soubor neexistuje, vytvoří se.

## Vstup dokumentu ze stejného zdroje jako příkazy

příkaz `n<<ukončení`

- 'ukončení' je symbol konce vstupu (řetězec označující konec vstupu)
- Řetězec 'n<<ukončení' musí být bez mezer.

```
mail novak <<EOF
```

```
První řádek dopisu
```

```
Druhý řádek dopisu
```

```
EOF
```

## Vstup dokumentu ze stejného zdroje jako příkazy - pokračování

- Ve variantě 'n<< –ukončení' se vypouštějí všechny znaky TAB na začátku čtených řádků.

```
mail novak <<-EOF
```

```
    První řádek dopisu
```

```
    Druhý řádek dopisu
```

```
EOF
```

## Použití souboru spojeného s jiným deskriptorem:

```
n>&m
```

Použije se soubor spojený s deskriptorem 'm' i pro výstup deskriptoru 'n'. 'n' je implicitně 1 (stand. výstup). Příklad:

```
ls neni je > dirlist 2>&1
```

Deskriptoru 1 se přiřadí soubor dirlist a deskriptor 2 se přiřadí k 1.

Není-li uvedeno 'n' a 'm', potom >& slouží jako operátor přesměrování stand. výstupu a současně stand. chybového výstupu do souboru:

```
ls neni je >& vse
```

## Přesměrování stand výstupu na chybový:

```
echo Chybové hlášení >&2
```

Totéž, ale pro vstup:

```
n<&m
```

Použije se soubor spojený s deskriptorem 'm' i pro vstup z deskriptoru 'n'.

'n' je implicitně 0 (stand. vstup).

Zavření vstupu nebo výstupu:

```
n<&-
```

```
n>&-
```

Otevření deskriptoru pro vstup i výstup:

```
n<>soubor
```

# Spojování příkazů do kolon

## příkaz1 | příkaz2 [ | příkaz3 ... ]

- Operátor |propojí standardní výstup příkazu1 se standardním vstupem příkazu2 **rourou**.
- Příkazy takto propojené nazýváme **kolonou**.
- Ukončovacím kódem kolony je ukončovací kód posledního příkazu v koloně.

- Zápisem  
**! příkaz1 | příkaz2 [ | příkaz3 ... ]**  
se ukončovací kód kolony neguje.
- Negace ukončovacího kódu znamená:
  - 0 se mění na 1
  - nenulová hodnota se mění na 0



- Zápisem

**příkaz1** 2 > &1 | **příkaz2**

se na standardní vstup příkazu2 předá jak standardní výstup, tak i stand. výstup příkazu1 (v tomto případě nelze deskriptor za & vypustit).

# Spuštění příkazu na pozadí

## příkaz1 & [ příkaz2 & ...]

- Shell příkaz spustí asynchronně, tj. nečeká na dokončení.
- Stand. vstup takto spuštěného procesu je napojen na `/dev/null`
- Na takto spuštěný proces nebo procesy se nevztahuje znak **intr**.

# Job Control

Jak dostat úlohu pod správu Job Control?

- 1 Spustíme-li úlohu na pozadí (&), vypíše se identifikace:

```
ping aisa > /dev/null &
```

```
[1] 1234
```

Číslo úlohy je 1 a má top-level proces 1234.

- 2 Běží-li úloha normálně spuštěná na popředí a chceme s ní nějak naložit, stiskneme znak **susp**. Shell pošle procesu signál **STOP** a vypíše se:

```
ping aisa > /dev/null
```

```
^Z
```

```
[2]+ Stopped      ping aisa > /dev/null
```

# Příkazy:

## jobs

Vypíše seznam řízených úloh

```
[1]- Running          ping aisa > /dev/null &  
[2]+ Stopped         ping aisa > /dev/null  
'+' znamená current job, '-' znamená previous job.
```

## jobs

Na úlohu se lze odkazovat několika způsoby:

`%+` nebo `%%`

current job

`%-` previous job

`%n` určená úloha, kde n je její číslo z [n]

`%řetězec`

úloha, jejíž spouštěcí příkaz začíná řetězcem

`%?řetězec`

úloha, jejíž libovolná část spouštěcího příkazu obsahuje řetězec

`fg [job]`

Úloha se spustí na popředí

Není-li zadán job, potom se použije current job.

`bg [job]`

Totéž, ale na pozadí.

`kill [signál job ...]`

Pošle signál (implicitně TERM) úloze.

`wait [job]`

Zahájí čekání na dokončení úlohy a převezme její ukončovací kód.

Není-li job zadán, čeká se na ukončení všech úloh a návratový kód bude 0.

# Seznamy

- Seznamem rozumíme posloupnost žádného nebo více příkazů oddělených novým řádkem, středníkem (;), nebo ampersandem (&).
- Shell příkazy provádí v pořadí, v jakém jsou zapsány.
- Pokud zápis končí ampersandem (&), ihned se zahájí provádění následujícího.
- Návratový kód seznamu je návratový kód posledního prováděného procesu.

```
cd adresář ; ls
```

```
cd adresář;ls
```

```
cd adresář ; rm *
```

- Příkazy lze oddělovat operátory `&&` a `||`.
- Oddělovač `&&` znamená, že se druhý příkaz provede **pouze** tehdy, pokud ukončovací kód předchozího byl 0.
- Oddělovač `||` znamená, že se druhý příkaz provede **pouze** tehdy, pokud ukončovací kód předchozího byl nenulový.

```
cd adresář&&ls
```



- Operátory `&&` a `||` mají stejnou prioritu.
- Nižší prioritu mají operátory `;` & nový řádek.

```
cd adresář && ls ; touch soubor
```

```
cd adresář && ls && touch soubor
```

```
ls adresář 2>/dev/null || mkdir adresář
```

# Substituce příkazů

``příkaz``

`$(příkaz)`

- Operátor obrácený apostrof
- Určeno k zahrnutí standardního výstupu příkazu do příkazového řádku. Shell zadaný příkaz provede ve vnořeném shellu (subshell).
- Případné znaky "nový řádek" se nahrazují mezerou.

```
ls `pwd`
```

```
grep -i 'moje jméno' `cat /tmp/které_soubory`
```

# Proměnné

- Deklarace a přiřazení hodnoty proměnné:  
jméno=[hodnota]
- Hodnotou je řetězec (i prázdný) podléhající všem dosud popsaným expanzím (nepodléhá '\*' expanzi, viz dále).
- Použití proměnné:  
\$jméno  
\${jméno}

```
ADRESAR=/home/novak/vyuka/unix
```

```
ls $ADRESAR
```

```
mkdir $ADRESAR/projekt
```

```
ls ${ADRESAR}
```

```
mkdir ${ADRESAR}NEW
```

- Výpis nastavených proměnných:

```
set
```

```
set|more
```

- Podléhá všem expanzím:

```
$ TEXT=`pwd`;date  
Čt dub 12 13:57:59 CEST 2007  
$ set|grep TEXT  
TEXT=/home/brandejs/smaz  
$
```

- vyjma '\*' expanze:

```
$ ls  
a b c  
$ PROMENNA=*  
$ set|grep PROMENNA  
PROMENNA='*'  
$ echo $PROMENNA  
a b c  
$
```

## Zrušení proměnných:

```
unset proměnná ...
```

## Označení proměnných určených k exportování do potomků:

```
export proměnná ...
```

V některých shellech lze současně nastavovat a exportovat:

```
export proměnná=[hodnota]
```

## Načítání obsahu proměnných ze stand. vstupu:

```
read proměnná ...
```

Příkaz čte ze standardního vstupu slova a přiřazuje je do proměnných. Při čtení se provedou všechny expanze vyjma '\*' .

## Příklad:

```
echo Zadej adresář a soubor, \  
které se mají vytvořit:  
read ADRESAR SOUBOR  
mkdir $ADRESAR 2>/dev/null|| \  
    echo Chyba při vytváření $ADRESAR >&2  
touch $ADRESAR/$SOUBOR
```

## Některé proměnné užívané shellem

### HOME

Domovský adresář pro příkaz "cd bez parametru" apod.

### PWD

Běžný (pracovní) adresář; **pwd** (echo \$PWD).

### OLDPWD

Předchozí běžný adresář; **cd -**

### MAIL

Jméno souboru s poštovní schránkou uživatele.

### PATH

Seznam adresářů prohledávaných při spouštění souboru.



## MANPATH

Seznam adresářů prohledávaných příkazem **man**.

### PS1

Řetězec primárního promptu (implicitně \$).

### PS2

Řetězec sekundárního promptu (implicitně >), tj. prompt pokračovacích řádků.

### IFS

Internal Field Separator – oddělovače polí v příkazu. Proměnná normálně obsahuje: mezeru, tabulátor, nový řádek.

Další viz "man sh" (ksh, bash, ... příslušný shell)

# Závorkování

(seznam)

- Posloupnost příkazů 'seznam' se provede ve vnořeném shellu.
- Tzn. že proměnné nastavené uvnitř závorek se po opuštění pravé závorky ztrácejí.
- Návrátový kód je návratový kód seznamu.

{ seznam;}

- Posloupnost příkazů 'seznam' se provede v aktuálním shellu.

**Příklad:**

```
(pwd; cd nic; pwd); pwd
```

```
{ pwd;cd nic;pwd;};pwd
```

```
cd nic 2>/dev/null||{ mkdir nic;cd nic;}
```

# Složené příkazy

```
for jméno [ in slovo; ] do seznam; done
```

Příkaz expanduje 'slovo' a postupně vytvářené položky přiřazuje proměnné 'jméno' a provádí posloupnost příkazů 'seznam'.

Není-li 'in slovo;' zadáno, potom se 'seznam' provede pro každý poziční parametr (viz dále).

Příklady:

```
for JMENO in *; do echo $JMENO; done
for POLOZKA in a b c d
do
  echo $POLOZKA
done
```

```
case slovo in  
vzorek [ | vzorek ... ]) seznam;;  
...  
esac
```

Příkaz `case` expanduje 'slovo' a hledá je mezi zadanými vzorky s použitím expanzních znaků pro jména souborů (viz dále). Jakmile se shoduje, provede se seznam příkazů. Po nalezení první shody se dále už nehledá.

Příklad:

```
echo Zadej akci a soubor:
```

```
read AKCE SOUBOR
```

```
case $AKCE in
```

```
    smaz|remove|delete)  rm $SOUBOR;;
```

```
    vytvor|create)       touch $SOUBOR;
```

```
                        chmod 777 $SOUBOR;;
```

```
esac
```

```
if seznam; then seznam; [ elif seznam; then seznam; ] ...  
[ else seznam; ] fi
```

Provede se '**if** seznam;'. Pokud jeho návratový kód je nulový (OK), provede se '**then** seznam;'. Jinak se provede '**elif** seznam;' (...) nebo '**else** seznam;'.  
Návratový kód je kód posledního provedeného procesu nebo 0, pokud se neprovedl žádný příkaz.

**while** seznam; **do** seznam; **done**  
**until** seznam; **do** seznam; **done**

Seznam '**do** seznam;' se provádí tak dlouho, dokud je návratový kód

*'while seznam;' nulový*

*'until seznam;' nenulový*



**break** [n]

Ukončí n-tou úroveň cyklu **for**, **while**, **until**.

**continue** [n]

Zahájí další iteraci cyklu.

**true**

Návratový kód vždy 0.

## false

Návratový kód vždy 1.

Příklad:

```
while true; do
    echo y
done
```

# Příkaz test

```
test výraz  
[ výraz ]
```

Příkaz vyhodnotí výraz a nastaví návratový kód 0 (true) nebo 1 (false).

# Testování typu souboru

- b** *soubor*      *soubor* existuje a je blokovým speciálním souborem.
- c** *soubor*      *soubor* existuje a je znakovým speciálním souborem.
- d** *soubor*      *soubor* existuje a je adresářem.
- e** *soubor*      *soubor* existuje (pouze vestavěný příkaz **test**).
- f** *soubor*      *soubor* existuje a je normálním souborem.

- h** *soubor* nebo -**L** *soubor*  
*soubor* existuje a je symbolickým odkazem.
- p** *soubor*  
*soubor* existuje a je pojmenovanou rourou.
- S** *soubor*  
*soubor* existuje a je socket.
- t** [*fd*]  
*fd* je otevřeno na terminál. Vynecháme-li *fd*, potom se použije hodnota 1 (standardní výstup).

# Testování přístupových práv

**-g** *soubor*

*soubor* existuje a má nastaven bit SGID.

**-k** *soubor*

*soubor* existuje a má nastaven 'sticky' bit.

**-r** *soubor*

*soubor* existuje a lze jej číst.

**-u** *soubor*

*soubor* existuje a má nastaven bit SUID.

**-w** *soubor*

*soubor* existuje a lze do něj zapisovat.

**-x** *soubor*

*soubor* existuje a je proveditelný.

# Testování charakteristik souborů

- e** *soubor*                      *soubor* existuje.
- s** *soubor*                      *soubor* existuje a má velikost větší než nula.
- soubor*<sub>1</sub> **-nt** *soubor*<sub>2</sub>                      *soubor*<sub>1</sub> je novější (podle času poslední modifikace obsahu) než *soubor*<sub>2</sub>.
- soubor*<sub>1</sub> **-ot** *soubor*<sub>2</sub>                      *soubor*<sub>1</sub> je starší (podle času poslední modifikace obsahu) než *soubor*<sub>2</sub>.
- soubor*<sub>1</sub> **-ef** *soubor*<sub>2</sub>                      *soubor*<sub>1</sub> a *soubor*<sub>2</sub> jsou na stejném zařízení a mají stejné číslo i-uzlu (tzn. jde o dva tvrdé odkazy na tentýž soubor).

# Testování řetězců

-z řetězec                      Délka řetězce je nulová

-n řetězec nebo řetězec                      Délka řetězce je nenulová.

řetězec<sub>1</sub> = řetězec<sub>2</sub>                      Řetězce jsou shodné.

řetězec<sub>1</sub> != řetězec<sub>2</sub>                      Řetězce se neshodují.



# Numerické testy

Na místě numerického argumentu se smí vyskytovat číslo(může být i záporné).

Numerické relační výrazy se vytvářejí následovně:

*číslo*<sub>1</sub> **-eq** *číslo*<sub>2</sub>

$$\textit{číslo}_1 = \textit{číslo}_2$$

*číslo*<sub>1</sub> **-ne** *číslo*<sub>2</sub>

$$\textit{číslo}_1 \neq \textit{číslo}_2$$

*číslo*<sub>1</sub> **-lt** *číslo*<sub>2</sub>

$$\textit{číslo}_1 < \textit{číslo}_2$$

*číslo*<sub>1</sub> **-le** *číslo*<sub>2</sub>

*číslo*<sub>1</sub> ≤ *číslo*<sub>2</sub>

*číslo*<sub>1</sub> **-gt** *číslo*<sub>2</sub>

*číslo*<sub>1</sub> > *číslo*<sub>2</sub>

*číslo*<sub>1</sub> **-ge** *číslo*<sub>2</sub>

*číslo*<sub>1</sub> ≥ *číslo*<sub>2</sub>

POKUS=abc test  $\${\#POKUS}$  -gt 1 && echo ano  
 $\${\#POKUS}$  je délka řetězce ve znacích

# Logické výrazy

**!** výraz

výraz je nepravdivý.

výraz<sub>1</sub> **-a** výraz<sub>2</sub>

výraz<sub>1</sub> a výraz<sub>2</sub> jsou pravdivé.

výraz<sub>1</sub> **-o** výraz<sub>2</sub>

výraz<sub>1</sub> nebo výraz<sub>2</sub> je pravdivý.

Priority můžeme upravovat i závorkami ( ); nesmíme však zapomenout závorky odstínit od shellovských expanzí např. \ ( \)

# Shell skript – scénář

Soubor s příkazy pro shell.

Soubor 'skript' lze číst

- Přístupové právo 'r'
- Spuštění ve vnořeném shellu:  
bash skript  
Soubor 'skript' se hledá v běžném adresáři a \$PATH
- Spuštění v běžném shellu:  
. skript  
Souboru 'skript' se hledá v \$PATH a v běžném adresáři.

## Soubor 'skript' lze číst a provést

- Přístupová práva 'rx'
- Spuštění ve vnořeném shellu:  
skript  
Soubor 'skript' se hledá v \$PATH

Shell skript se provede vždy tím programem (shellem), ze kterého byl spuštěn.

Program, kterým se shell skript má provést, lze specifikovat na prvním řádku od prvního sloupce:

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

- Soubor se shell skriptem **musí** být vždy čitelný (r).
- Shell skript **nelze** SUID, SGID.

Komentáře (poznámky) lze psát za znak '#' (znak je buď na začátku řádku, nebo jej bezprostředně předchází bílé místo).

# Poziční parametry

Poziční parametr je pojmenován číslem větším než 0.

**\$1 \$2 \$3 ... \$9**

**\${1} \${2} \${3} ... \${9} \${10} \${11} ...**

Shell poziční parametry implicitně nastavuje na hodnoty argumentů uvedených na spouštěcím řádku.

Příklad:

```
skript a b c
```

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

```
echo $1 $2 $9 ${10}
```

Explicitně se poziční parametry nastavují příkazem **set** následovně:

```
set -- a b c  
echo $1 $2 $3
```

### **shift** [*n*]

*Příkaz "posune" obsahy pozičních parametrů o n nebo jeden doleva. Počet pozičních parametrů (\$#) se sníží o jeden.*



# Zvláštní parametry

`$*` Expanduje se do pozičních parametrů od prvního do posledního zadaného. Je-li parametr uveden uvnitř dvojice úvozovek, expanduje se do jednoho slova tak, že se parametry oddělí prvním znakem uvedeným v IFS (nebo mezerou, není-li IFS).

```
skript 1 2 '3 4' 5
```

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

```
IFS=!
```

```
echo "$*"# Vypíše 1!2!3 4!5
```

`$#` Expanduje se do počtu pozičních parametrů.

```
#!/bin/sh
if [ $# = 0 ]; then
    echo Příkaz se spouští s těmito \
    parametry ... >&2
    exit 1
fi
```

**\$?** Expanduje se do návratového kódu posledního dokončeného procesu.

```
#!/bin/sh
mkdir adresar 2> /dev/null
if [ $? != 0 ]; then
    echo Chyba ... >&2
fi
```

- \$- Expanduje se do řetězce znaků zadaných jako volby.
- \$\$ Expanduje se do čísla procesu shellu, který provádí expanzi (viz typický příklad na vytvoření pomocného souboru jedinečného jména).

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

```
TEMP=/tmp/pomocny.$$
```

```
touch $TEMP
```

```
...
```

```
rm $TEMP
```

**\$!** Expanduje se do čísla nejposlednějšího procesu spuštěného na pozadí.

```
ping aisa &  
sleep 2  
kill $!
```

**\$0** Expanduje se do uživatelem zadaného jména skriptu.

# Expanze proměnných

`${proměnná}`

Nejjednodušší tvar.

`${proměnná:-slovo}`

Použij implicitní hodnotu.

Pokud je proměnná nedefinována nebo prázdná, použije se expandované slovo.

`${proměnná:=slovo}`

Přiřaď implicitní hodnotu.

Pokud je proměnná nedefinována nebo prázdná, přiřadí se a použije se expandované slovo.

`${proměnná:?slovo}`

Oznam chybu.

Pokud je proměnná nedefinována nebo prázdná, předá se na stand. chybový výstup hlášení slovo.

`${proměnná:+slovo}`

Použij jinou hodnotu.

Pokud je proměnná nedefinována nebo prázdná, předá se prázdná hodnota. Je-li neprázdná, použije se slovo.

Vypustí-li se dvojtečka, testuje se pouze nedefinovanost.

`${#proměnná}`

Délka řetězce proměnné ve znacích.

# Expanze proměnných (řetězce)

$\${proměnná\%slovo}$

Odstraní nejkratší sufix (podřetězec končící s koncem řetězce).

Zadané *slovo* se expanduje do podoby vzorku na obsahu *proměnné*. Pokud operátor najde vzorek na konci řetězce, odstraní jeho nejkratší vyhovující část.

$\${proměnná%%slovo}$

Odstraní nejdelší sufix.

Zadané *slovo* se expanduje do podoby vzorku na obsahu *proměnné*. Pokud operátor najde vzorek na konci řetězce, odstraní jeho nejdelší vyhovující část.



### $\${proměnná\#slovo}$

Odstraní nejkratší prefix (podřetězec začínající se začátkem řetězce).

Zadané *slovo* se expanduje do podoby vzorku na obsahu *proměnné*. Pokud operátor najde vzorek na začátku řetězce, odstraní jeho nejkratší vyhovující část.

### $\${proměnná##slovo}$

Odstraní nejdelší prefix.

Zadané *slovo* se expanduje do podoby vzorku na obsahu *proměnné*. Pokud operátor najde vzorek na začátku řetězce, odstraní jeho nejdelší vyhovující část.

# Vlnková expanze

`~novak`

Domovský adresář uživatele novak (např. /home/novak).

`~novak/pub`

Např. /home/novak/pub

`~`

Domovský adresář běžného uživatele (podle \$HOME).

~/.profile

Soubor .profile v domovském adresáři běžného uživatele.

~+

Běžný adresář (podle \$PWD).

~-

Předchozí běžný adresář (podle \$OLDPWD).

## \* apod. expanze

\*

Vyhovuje libovolnému počtu libovolných znaků.

?

Vyhovuje právě jednomu libovolnému znaku.

[ ]

Definuje třídu vyhovujících znaků; obsah hranatých závorek popisuje právě jeden znak.

[ACd-i]\*

Vyhovuje jménům začínajícím znakem A, C, d, e, f, ..., i.

[!0-9]\*

Vyhovuje jménům začínajícím čímkoli vyjma číslic. Smysl obrací operátor ! (starší shelly používaly ^).

# Expanze jmen souborů a adresářů

- Vzorek nesmí obsahovat lomítko, tzn. do jednoho vzorku lze zahrnout nejvýše jeden adresář.  
Příklad: `/usr/s*/*`
- Vzorek nevyhovuje jménům adresářových položek začínajících tečkou. Nutno zadat explicitně.  
Příklad: `.* *`
- Každé expadované slovo se nahradí posloupností vyhovujících jmen oddělených mezerou.  
Příklad: `.[!..]*`  
Vše mimo `..` (a položek, jejichž jméno začíná `..`).

# Funkce v shellu

Syntax definice:

```
jméno() { seznam;}
```

Volání:

```
jméno [argumenty]
```

Příklad:

```
datum()  
{  
    date '+%-d. %-m. %Y %H:%M'  
}  
datum
```

## Proměnné globální a lokální:

**Globální:** nastavovány klasickým způsobem

**Lokální:**

**local** *proměnná*[=...]

- **local** je interní příkaz shellu.
- Lze použít pouze uvnitř funkce.
- Deklaruje-li se lokální proměnná stejného jména jako globální, lokální zdědí počáteční hodnotu z globální.
- Hodnota nastavená lokální proměnné uvnitř funkce nezmění globální hodnotu po opuštění funkce.
- Lokální proměnná se dědí do vnořených funkcí.



- Poziční parametry se nastaví tak, jak byly zadány při spuštění funkce (vyjma \$0).

Příklad: Vyzkoušejte si

```
fce() {  
    echo $0  
    echo $*  
}  
fce a b c d
```

- Příkaz: **return** [*návratový kód*]
  - Interní příkaz shellu.
  - Ukončí právě prováděnou funkci.
- Funkce se provádějí v běžném shellu.
- Funkce mohou být rekurzivní (bez omezení počtu vnoření).
- Funkce se nedědí.

# Aritmetické expanze

**`$((výraz))`**

Alternativním způsobem zápisu v některých shellech (např. bash) je:

**`$(výraz)`**

Výraz se interpretuje podle těchto pravidel:

- 1 výraz se interpretuje tak, jako by byl uzavřen do uvozovek (uvozovky uvnitř výrazu nemají řídicí význam),
- 2 ve výrazu se expandují parametry (`$...`) a substituce příkazů (`'...'`),
- 3 příkaz se vyčíslí.

Výraz se vyčísluje v aritmetice **long integer** bez kontroly přetečení.

# Aritmetické expanze

Jako příklad si uveďme skript, který spočítá počet podadresářů v běžném adresáři:

```
#!/bin/sh
POM=0
for JMENO in *
do
    if [ -d $JMENO ]; then
        POM=$((POM+1))
    fi
done
echo V adresari `pwd` je $POM podadresaru.
```

# Operátory (zapsány s klesající prioritou)

- +	unární minus a plus
! ~	logická negace a inverze bitů
* / %	násobení, dělení, zbytek po dělení
+ -	sečítání, odčítání
<< >>	posun bitů vlevo a vpravo ( <i>co « o kolik bitů</i> )
<= >= < >	porovnávání
== !=	rovnost, nerovnost
&	součin (AND) po bitech
^	exkluzivní součet (XOR – nonekvivalence) po bitech
	součet (OR) po bitech
&&	logický součin
	logický součet
= *= /= %= += -= <<= >>= &= ^	operace s přiřazením výsledku

Operandy s operátory lze uzavírat do závorek ( ). Závorky pak mají nejvyšší prioritu.

Proměnnou lze expandovat i na operátor.

### Konstanty:

<i>Formát zápisu</i>	<i>Konstanta n je v</i>
$0n$	osmičkové soustavě
$0xn$	šestnáctkové soustavě (nebo též $0Xn$ )
$báze\#n$	soustavě o základu <i>báze</i> , kde <i>báze</i> je číslo desítkové v intervalu 2 až 36
$n$	desítkové soustavě

Konstanta se vždy převádí do desítkové soustavy.

## Pořadí vyhodnocování expanzí

- 1 ~ expanze, expanze parametrů, substituce příkazů ( ' ' ), aritmetické expanze (všechny tyto expanze se vyhodnocují na stejné úrovni),
- 2 oddělení polí generovaných v kroku 1 (pokud je IFS neprázdné),
- 3 \* apod. expanze (jména souborů a adresářů), nebyla-li při spuštění shellu uvedena volba **-f**,
- 4 odstranění apostrofů a uvozovek.

# Nalezení a provedení příkazu

Shell rozlišuje tři různé typy příkazů:

- shellovské funkce,
- interní příkazy a
- normální programy.

Shell příkazy rovněž hledá v tomto pořadí:

- 1 Po zadání příkazu prohledá seznam shellovských funkcí,
- 2 potom seznam interních příkazů a
- 3 nakonec adresáře v seznamu PATH.



# Volba --

```
touch -- -l  
ls  
rm -- -l
```

# Interní příkazy shellu

```
. soubor [argumenty]  
source soubor [argumenty]
```

Příkaz přečte a provede obsah souboru *soubor* v rámci aktuálního shellu.

Návratový kód posledního příkazu provedeného uvnitř souboru se stane návratovým kódem příkazu (nebo se vrátí 0, pokud nebyl proveden žádný příkaz).

Pokud slovo *soubor* neobsahuje lomítko, potom se jméno *soubor* hledá podle seznamu v PATH. Soubor nalezený podle PATH se provede, i když nebude proveditelný. Pokud se *soubor* nenajde, je návratový kód 1.

Volitelné argumenty se spuštěnému souboru předají jako poziční parametry. Pokud argumenty nejsou zadány, potom se poziční parametry nemění.

## **alias** [*přezdívka*[=*řetězec*] ...]

Příkazem se definují přezdívky pro řetězce. Řetězcem může být příkaz nebo příkaz vč. argumentů, např.

`alias l='ls -l'` Je-li uvedena přezdívka bez řetězce, vypíše se její aktuální nastavení.

Příkaz **alias** bez argumentů vypíše aktuální nastavení všech přezdívek.

Návratový kód příkaz vrací nulový vyjma případu, kdy uživatel zadá k výpisu obsah neexistující přezdívky. Viz též **unalias**.

## bg [úloha] ...

Přesune provádění určené úlohy na pozadí tak, jako by byla spuštěna oddělovačem příkazů **&**. Pokud není zadána úloha, potom se příkaz odkazuje na *běžnou* úlohu (viz příkaz **job**).

Příkaz vrací návratový kód 0 vyjma následujících případů: job control je vypnutý, úloha neexistuje nebo byla spuštěna bez job control. Viz též **fg**, **kill**.

# Interní příkazy shellu

**builtin** *interní\_příkaz* [*argumenty*]

Příkaz provede zadaný interní příkaz shellu, případně mu předá zadané argumenty a převezme návratový kód.

Tento příkaz je užitečný v případě, že jsme si definovali funkci shellu, která má jméno shodné s některým z interních příkazů.

## cd [*adresář*|-]

Příkaz změní běžný adresář na *adresář*.

Změna se mj. projeví v novém nastavení obsahu proměnné PWD.

Uživatel si může definovat proměnnou prostředí CDPATH a nastavit do ní seznam cest k adresářům, ve kterých se bude hledat argument *adresář*. Jednotlivé cesty se oddělují dvojtečnou (:).

Pokud *adresář* začíná lomítkem (/), potom se seznam cest v CDPATH ignoruje.

Příkaz **cd** vedle proměnné PWD také nastavuje proměnnou OLDPWD. V této proměnné příkaz uchovává cestu k předchozímu běžnému adresáři.

Příkaz **cd** proměnnou OLDPWD použije tehdy, když jako argument zadáme minus (-). V tomto případě se jako běžný adresář nastaví ten, který byl běžný před aktuálním běžným adresářem.

**command** [-pVv] *příkaz* [*argumenty*]

Zadaný *příkaz* se spustí bez prohledávání seznamu shellovských funkcí.

# Interní příkazy shellu

## echo [-neE] [*argumenty*]

Příkaz vypíše na standardní výstup argumenty oddělené mezerou.

Návratový kód je vždy 0.

Po uvedení volby **-n** se potlačí výstup nového řádku za posledním argumentem.

Volba **-e** zapíná řídicí význam následujících sekvencí. Volba **-E** tento význam vypíná (u systémů, které je implicitně interpretují jako řídicí).



<code>\a</code>	pípne (alert)
<code>\b</code>	o znak vzad (backspace)
<code>\c</code>	potlačí výstup nového řádku
<code>\f</code>	nová stránka (form feed)
<code>\n</code>	nový řádek (new line)
<code>\r</code>	návrat na začátek řádku (cr)
<code>\t</code>	horizontální tabulátor
<code>\v</code>	vertikální tabulátor
<code>\\</code>	obrácené lomítko
<code>\nnn</code>	osmičkový kód ASCII znaku

```
echo -e 'Dnešní datum: \c'; date
```

## **eval** [*argumenty*]

Příkaz přečte, expanduje a spojí *argumenty* do jednoho příkazu. Takto vytvořený příkaz se předá shellu k expanzi a provedení.

Návratový kód provedeného příkazu se předá jako návratový kód příkazu **eval**. Při zadání prázdného nebo žádného argumentu je návratový kód nulový.

Následující příklad vypíše 'C':

```
A=B; B=C
```

```
eval echo \$$A
```

# Interní příkazy shellu

```
exec [[ - ] příkaz [argumenty]
```

Existuje-li *příkaz*, spustí se nahrazením shellu. Nevytváří se žádný nový proces. Uvedené *argumenty* se předají spuštěnému *příkazu*.

Je-li uvedeno minus (-), potom se jméno nového příkazu v parametru \$0 předradí minus. Toto login automaticky provádí u prvního spuštěného procesu.

Pokud *příkaz* není zadán, uplatní se pouze případná přesměrování a návratový kód se nastaví na 0.

## exit [*návratový\_kód*]

Příkaz ukončí shell a předá *návratový kód*. Pokud operand zadán není, předá se návratový kód posledního ukončeného procesu.

## **export** [*jméno ...* ]

Příkaz uvedené proměnné označí jako exportovatelné do procesů, které budou aktuálním shellem spouštěny. Takto označené proměnné synovské procesy dědí, ostatní ne.

Pokud se v příkazu žádný argument neuvede, vypíše se ty proměnné, které již takto označeny jsou.

Shellové funkce se nedědí ani po označení tímto příkazem.

## fg [úloha] ...

Přesune provádění určené úlohy na popředí.

Pokud není zadána úloha, potom se příkaz odkazuje na *běžnou* úlohu (viz příkaz **job**).

Příkaz vrací návratový kód takový, jaký předá ukončený proces (ten, co byl přesunut na popředí), nebo vrátí chybový návratový kód pokud: job control je vypnutý, úloha neexistuje nebo byla spuštěna bez job control. Viz též **bg**, **kill**.

# Interní příkazy shellu

**getopts** *povolené\_volby jméno [argumenty]*

```
#!/bin/sh
while getopts abo: VOLBA
do
    case $VOLBA in
    a|b)    FLAGS=$FLAGS$VOLBA;;
    o)     OARG=$OPTARG;;
    \?)    echo Pouziti ... 1>&2
           exit 1;;
    esac
done
echo $*
shift $(( $OPTIND - 1 ))
echo $*
```

Při každém volání **getopts** se předá jedna volba z příkazového řádku do proměnné *jméno* a index volby následující se uloží do proměnné OPTIND. Při každém spuštění shellu nebo skriptu se proměnná OPTIND nastavuje na hodnotu 1. Tento skript lze spustit např. jedním z následujících způsobů:

```
skript -a -b -o "xxx z yy"soubor  
skript -ab -o "xxx z xx"-- soubor
```

Příkaz **getopts** implicitně zpracovává volby, které byly zadány na příkazovém řádku. Explicitně však volby můžeme zadat i na řádku s **getopts** jako *argumenty*. V tomto případě se použijí *argumenty* místo obsahu příkazového řádku.



## hash [-r] [jméno]

Shell si vytváří pomocný seznam jmen příkazů s již zjištěnou absolutní cestou.

Příkazem **hash** bez uvedení parametru vypíšeme aktuální verzi seznamu.

## help [vzorek]

Příkaz zobrazí nápovědu buď k interním příkazům zadaným vzorkem, nebo vypíše seznam všech interních příkazů. Návratový kód je nula vyjma případu, kdy zadanému vzorku nevyhovoval žádný příkaz.

## jobs [-lnp] [úloha ...]

Příkaz vypíše všechny aktivní úlohy. Po uvedení volby **-l** se zahrnou i čísla procesů, které tvoří úlohy. Volba **-p** zapíná výpis pouze čísel prvních procesů jednotlivých úloh. Zadáním volby **-n** požadujete výpis pouze změn od předchozího spuštění příkazu.

```
kill [-signál] pid|úloha ...  
kill -l
```

Příkaz předá procesu (zadaným číslem *pid*) nebo úloze (určené kódem *úloha*) signál.

## **let** *argument ...*

Příkaz vyčíslí každý ze zadaných aritmetických výrazů. Je-li výsledek posledního aritmetického výrazu nenulový, potom je návratový kód příkazu **let** nulový. V opačném případě se vrací jednička.

**local** [*jméno*[=*hodnota*]]

Příkaz definuje lokální proměnnou a přiřadí jí počáteční hodnotu. Příkaz lze použít pouze uvnitř funkce.

## pwd

Příkaz vypíše absolutní cestu k právě nastavenému běžnému adresáři.

# Interní příkazy shellu

**read** [*jméno* ...]

Příkaz přečte ze standardního vstupu jeden řádek a první přečtené slovo přiřadí prvnímu *jménu* proměnné, druhé slovo druhé proměnné atd.

Pokud se neuvedou žádná *jména*, potom se celý přečtený řádek přiřadí do proměnné REPLY.

Návratový kód je nulový vyjma případu, kdy se načte konec souboru (^D).



**readonly** [*jméno*[=*hodnota*] ... ]

Příkazem označené proměnné lze od okamžiku zadání tohoto příkazu pouze číst.

## return [*n*]

Příkaz ukončuje provádění funkce a vrátí řízení volajícímu. Návratový kód je buď *n*, nebo návratový kód posledního dokončeného příkazu.

Příkaz lze použít ve skriptu spuštěném příkazem '.' (**source**) se stejným významem. Při použití příkazu jinde se nastaví chybový návratový kód.

## set [-volby|+volby|--] [argumenty]

Bez parametrů příkaz vypisuje seznam proměnných a jejich aktuální obsah.

V aktuálním shellu mění nastavení *voleb*. Předponou '-' se volba nastavuje, předponou '+' se volba ruší. Volbou '--' se ukončuje zadávání voleb.

Zadané *argumenty* se interpretují jako nové hodnoty pozičních parametrů.

Potřebujeme-li zrušit poziční parametry, zadáme:

```
set --
```

Poziční parametry zrušíme také příkazem 'shift \$#'.

**shift** [*n*]

Příkaz "posune" poziční parametry o jednu nebo *n* pozic.

# Interní příkazy shellu

**trap** [*akce*] [*signál ...*]

Příkazem se definuje, jaká *akce* se má provést v případě, že shell přijme (rozpozná) zadaný *signál*.

Pokud se nezadá *akce*, nebo se na místě akce zadá '-', potom se chování pro zadané signály nastaví na implicitní (tak, jak je nastaveno po spuštění shellu).

Pokud je argument *akce* prázdný (tj. dva apostrofy bezprostředně za sebou), potom se signál ignoruje.

Zadáte-li akci pro signál 0 (EXIT), potom se tato akce provede při ukončování shellu (např. příkazem **exit**).

```
type jméno ...
```

Příkaz pro každé zadané *jméno* vypíše, jak se bude interpretovat (co se spustí).

## ulimit [-volba [limit]]

Příkazem se nastavují limitní hodnoty užívání zdrojů systému.

**Tvrký** (hard) limit (volba H) se smí nastavit pouze jednou (příp. jej smí nastavit pouze superuživatel), uživatel smí měnit hodnotu **měkkého** (soft) limitu (volba S) a to nejvýše na tvrdou hodnotu.

- a vypíše se všechny nastavené limity,
- c nastaví se maximální velikost paměťového obrazu jádra, které se ukládá na disku (core, core dump),
- d maximální velikost datového segmentu nebo haldy,
- f maximální velikost vytvářeného souboru,
- n maximální počet otevřených souborů,
- s maximální velikost segmentu se zásobníkem,
- t maximum času procesoru,
- v maximální velikost virtuální paměti.

# Interní příkazy shellu

## **umask** [*nnn*]

Příkazem se nastaví implicitní formát masky pro přidělování přístupových práv vytvářenému objektu (souboru, adresáři).

Operand se zadává zpravidla osmičkově a představuje ty bity přístupových práv, které se **nemají** nastavit.

Např. 'umask 022' znamená, že se pro skupinu a ostatní nenastaví právo w (právo zápisu).



## **unalias** [-a] [*jméno* ...]

Příkazem se odstraní zadané *jméno* (nebo jména) ze seznamu přezdivek (viz příkaz **alias**).

Volbou **-a** se odstraní všechny přezdívky.

**unset** *jméno* ...

Příkaz odstraní uvedené proměnné.

Odstranit nelze proměnné PATH, PS1, PS2, MAILCHECK a IFS (příp. i další).

## wait [*n*]

Příkaz čeká na ukončení specifikovaného procesu nebo úlohy.

Jakmile se proces ukončí, ukončí se i příkaz a od procesu převezme návratový kód a vrátí jej jako svůj návratový kód. Pokud zadaný proces neexistuje, vrací se 127.

Parametr *n* je buď číslo procesu, nebo kód úlohy. Bez zadání parametru příkaz čeká na dokončení všech v době zadání příkazu aktivních synovských procesů a vrátí nulový návratový kód.

# Spuštění shellu

- **Login shell** je ten interpret příkazů, který má jméno ve tvaru '*-jméno*', tzn. příkaz 'echo \$0' vrátí např. '-ksh'. Předpokládá se, že jde o první shell uživatele – shell, který se uživateli spustí při přihlášení.
- **Interaktivní shell** je ten, který má standardní vstup a standardní výstup připojen k terminálu nebo byl spuštěn s volbou **-i**.

# Spuštění login shellu

Po spuštění login shellu se nejprve hledá soubor

**`/etc/profile`**

Pokud existuje, provede se. Dále se hledá soubor

**`~/.profile`**

Pokud existuje, rovněž se provede.

## Spuštění ostatních shellů a pokračování login shellu

Po spuštění každého shellu (tedy i v login shellu po provedení `~/.profile`) se provádí skript, který je uveden v proměnné ENV.

Tuto operaci lze zapsat následovně:

```
if [ "$ENV" ]; then . $ENV; fi
```

Zpravidla se takový skript jmenuje `~/.kshrc`, `~/.bashrc` apod.

Do tohoto souboru zpravidla ukládáme definice vlastních shellovských funkcí (ty se jinak nedědí), definice aliasů apod.

# Volby shellu

- Při spuštění shellu na příkazovém řádku nebo
- interaktivně uvnitř aktuálního shellu příkazem **set**

## **-a allexport**

Automaticky označuje všechny proměnné příznakem exportování (viz **export**).

## **-b notify**

Zapíná okamžité ohlašování ukončení procesů spuštěných na pozadí (ne až před výpisem promptu).

## **-C noclobber**

Při přesměrování výstupu ( $>$ ) do existujícího souboru se tento nepřepíše.

**-e errexit**

Neinteraktivní shell se ukončí při zjištění prvního chybového návratového kódu, který není ve skriptu testován. Shell se neukončuje tehdy, pokud proces s chybným návratovým kódem řídí příkazy `if`, `elif`, `while`, `until`, nebo pokud je za procesem uveden operátor `&&` nebo `||` (nebo pokud se návratový kód neguje uvedením `!`).

**-f noglob**

Zakazuje se expanze jmen souborů a adresářů.

**-i interactive**

Násilně prohlásí shell za interaktivní.



**-m monitor**

Zapne job control (implicitní varianta v interaktivním shellu).

**-n noexec**

V neinteraktivní shellu příkazy pouze čte, ale neprovádí je. Volba je vhodná k testování syntaxe skriptů a ignoruje se v interaktivní shellu.

**-o jméno\_volby**

Za volbou **-o** se zadávají volby jménem, např. **-o allexport**. Kromě voleb uvedených výše a níže, lze zadat i některé další:

**ignoreeof**

Shell nelze ukončit stiskem ^D.

**-t**

Shell se ukončí po přečtení a provedení jednoho příkazu.

**-u nounset**

Shell vypíše na standardní chybový výstup hlášení při pokusu o expanzi nedefinované (unset) proměnné. Není-li shell interaktivní, tak se navíc ukončí s nenulovým návratovým kódem.

**-v verbose**

Shell opisuje vstup na standardní chybový výstup (užitečné pro ladění skriptů).

**-x xtrace**

Po expanzi každého příkazu jej před provedením vypíše na standardní chybový výstup s prefixem '+' (nebo s hodnotou uloženou v PS4).

## Volby zadávané pouze při spuštění shellu

Pokud není zadána volba **-c** ani **-s**, potom se první argument chápe jako jméno skriptu, který se má v rámci spouštěného shellu provést. Ostatní argumenty se stávají pozičními parametry.

### **-c** řetězec

Shell přečte příkazy z řetězce, provede je a skončí.

- s** Shell se spustí jako interaktivní. Příkazy se čtou ze standardního vstupu a hlášení shellu se posílají na standardní chybový výstup. Následují-li na příkazovém řádku argumenty, použijí se jako poziční parametry. Interaktivní shell se spustí také v případě, že na příkazovém řádku není žádný argument.

- i Shell se spustí jako interaktivní. Příkazy se čtou ze standardního vstupu a hlášení shellu se posílají na standardní chybový výstup.
- r Spustí shell v omezeném (restricted) režimu.

Ostatní volby, které zde lze také použít, jsou popsány pod heslem *Volby shellu*.

# Shell v omezeném režimu

Shell se do omezeného (restricted) režimu dostane

- buď spuštěním s volbou **-r**,
- nebo spuštění shellu jménem typu **rsh**, **rksh** apod.

Tato jména bývají zpravidla odkazy na sh, ksh apod.

Omezený shell se liší v následujících bodech:

- 1 je vypnuto přesměrovávání výstupu,
- 2 nelze použít interní příkaz **cd**,
- 3 nelze měnit proměnnou PATH,
- 4 jména začínající / nejsou akceptována,
- 5 proměnná SHELL se nastaví např. na `rsh` a nelze ji změnit.

Výše uvedená omezení vstupují v platnost až po provedení souboru `.profile`

# Editor vi – Visual Interactive

Spuštění:

**vi** *soubor*

**Nejdůležitější informace - Editor se ukončuje následovně:**

:

Ukončí obrazovkový režim a přejde do řádkového režimu.

q

Ukončí editor tehdy, nebyly-li provedeny změny textu.

q!

Ukončí editor bez uložení změn.

x

Ukončí editor s uložení změn do souboru *soubor*.

## Režimy editoru vi:

Rozlišujeme tyto základní režimy činnosti editoru:

- obrazovkový režim
  - režim zadávání příkazů (po spuštění)
  - vkládací režim
- řádkový režim

Do vkládacího režimu se z režimu zadávání příkazů dostaneme příkazy `a`, `i` apod.

Režim se ukončuje stiskem `ESC` a vrátíme se do režimu zadávání příkazů.

Do řádkového režimu vstoupíte po stisku `:`.



# Editor **ed**

- Standardní textový editor UNIXu.
- Pracuje pouze v řádkovém režimu (nelze obrazovkový režim).
- Editovaný soubor se při otevírání kopíruje do vnitřního bufferu editoru.
- Editor **ed** po celou dobu editace pracuje pouze s pracovní kopií, změna originálního souboru se provede až po zadání příkazu **w**.
- Editor **ed** v omezeném režimu se spouští příkazem **red** (editace pouze souborů v běžném adresáři, nelze ! apod.).
- Příkazy editoru **ed** lze použít jako příkazy řádkového režimu editoru **vi**.
- Editor **ed** podporuje pouze základní (omezené) regulární výrazy (rozšířené viz **egrep** a další).

- Použití regulárních výrazů:
  - pro určení řádku (vyhledáním podle vzorku)
  - náhrada řetězce (záměna vzorku za jiný)

```
ed diplomka.tex
1,$s/unix\[^a-z]\)/UNIX\1/g
w
q
```

Editor po načtení souboru do bufferu vypíše počet načtených znaků. Na chybu reaguje pouze výpisem otazníku.

# Jednoznakové regulární výrazy (RE):

1.1 Konkrétní znak vyhovuje sám sobě.

a vyhovuje a

1.2 Dvojice znaků obrácené lomítko a znak vyhovuje vždy znaku za obráceným lomítkem. Významné je v souvislosti s těmito zvláštními znaky:

- . \* [ \ jsou vždy zvláštními znaky
- ^ je zvláštním znakem na úplném začátku RE nebo v kombinaci s [^]
- \$ je zvláštním znakem na úplném konci RE
- Zvláštním znakem je znak sloužící k oddělení RE; např.  
s/RE/náhrada/

1.3 . vyhovuje každému znaku vyjma nového řádku.

1.4 [0-9]

[^0-9]

Použití zvláštních znaků:

[ ] a-b]

[-a-b]

[a^]

[[a]

## Konstrukce RE z jednoznakových RE:

- 2.1 Jednoznakový RE (JRE) je RE.
- 2.2 JRE bezprostředně následovaný hvězdičkou vyhovuje řetězci tvořenému žádným nebo více výskyty JRE.  
**Pozor:** Nepleťte si výraz `[a-d]*` zapsaný na příkazovém řádku shellu určený k expanzi s RE. Řetězec `[a-d]*` určený k shellovské expanzi odpovídá RE `[a-d].*`

Při více možnostech se vybere nejdelší možný levostranný výskyt (hladový přístup).

*Nejdelší možný:*

aaabbabaabbbbbaaaabab

s/[a-b]\*/xx/

xx

*Levostranný:*

ababa

s/aba/xx/

xxba

## 2.3

$JRE\{n\}$   $n$  výskytů  $JRE$

$JRE\{n, \}$   $\geq n$  výskytů  $JRE$

$JRE\{n, m\}$   $\geq n$  a  $\leq m$  výskytů  $JRE$

Je-li více možností, vybere se nejvyšší možný počet výskytů.

2.5 \`( \)`

2.6 \`n`

Příklady:

Michal Brandejs

```
s/^\([^\ ]*\)\ ([^\ ]*\)$/\2\1/
```

Brandejs Michal

echo Michal Brandejs| \`\`

```
sed 's/^\([^\ ]*\)\ ([^\ ]*\)$/\2 \1/'
```

```
^\(.*\)\1$
```

*Vyhovuje všemu, co obsahuje 2 opakující se řetězce na řádku.*



## Využití RE pro hledání slov:

- 3.1 `\ <` znamená začátek slova (číslice, písmena, podtržení)
- 3.2 `\ >` znamená konec slova

Příklad:

```
\ <a \ >
```

'a' pouze jako slovo

## Řetězce na začátku a konci řádku:

- 4.1  $\wedge$  na začátku RE znamená, že se hledá od začátku řádku
- 4.2  $\$$  na konci RE znamená, že se hledá na konci řádku
- 4.3  $\wedge \dots \$$  celý řádek

### Opakování RE:

- Prázdný RE se nahradí posledním neprázdným.

# Adresace řádků editoru ed

`.`  
běžný řádek

`$`  
poslední řádek editované pracovní kopie

`n`  
 $n$ -tý řádek (desítkově)

`'x`  
řádek označený značkou `x` (písmeno [a-z]; nastavuje se **kx**)

`/RE/`  
řádek vyhovující RE směrem vpřed

`?RE?`

řádek vyhovující RE směrem vzad

`cokoli+n`

od cokoli vpřed o  $n$  řádků

`cokoli-n`

od cokoli vzad o  $n$  řádků

`+n`

od běžného řádku o  $n$  řádků vpřed

`-n`

od běžného řádku o  $n$  řádků vzad

*cokoli+*

přičítá se 1

*cokoli-*

odčítá se 1

,

totéž co **1,\$**

;

totéž co **.,\$**

## Příkazy editoru ed

Na jednom příkazovém řádku smí vyskytnout nejvýše jeden příkaz.

Za některými příkazy (vyjma e, f, r a w) může být přípona l, n nebo p, která říká, jakým způsobem se má běžný řádek vypsát.

```
(.)a  
text  
.
```

Příkaz **append** vloží žádný nebo více řádků textu za adresovaný řádek pracovní kopie souboru.

```
(.)c  
text  
.
```

Příkaz **change** zruší v pracovní kopii souboru adresované řádky a na místo těchto řádků vloží žádný nebo více zadaných řádků.

```
(.,.)d
```

Příkaz **delete** zruší v pracovní kopii souboru adresované řádky.

## e *soubor*

Příkazem **edit** zrušíme celý obsah pracovní kopie souboru a načteme do ní jmenovaný *soubor*.

## E *soubor*

Příkaz **Edit** se chová podobně jako e s tím rozdílem, že nekontroluje, zda od posledního příkazu w byly v pracovní kopii souboru učiněny změny.

Zapamatuje se použité jméno souboru pro pozdější použití příkazy e, r a w.

Pokud jméno souboru se nahradí znakem vykřičník (!), potom se zbytek řádku předá shellu k provedení. Výstup z tohoto příkazu se načte.



f *soubor*

Příkaz **filename** změní zapamatované jméno na zadané jméno.

(1,\$)g/RE/*seznam příkazů*

Příkaz **global** označí všechny řádky, které vyhovují zadanému RE a potom pro každý označený řádek provede *seznam příkazů* tak, že zpracováváný označený řádek je nastaven jako běžný.

$(1, \$)G/RE/]$ 

Interaktivní příkaz **Global** nejprve označí všechny řádky, které vyhovují zadanému RE. Potom pro každý označený řádek provede tyto operace: vypíše jej, nastaví jej jako běžný a čeká na zadání **jednoho** příkazu (jiného než a, c, i, g, G, v a V).

h

Příkaz **help** vypíše stručné vysvětlení posledního stavu avizovaného výpisem znaku ?.

H

Příkazem **Help** se editor přepíná do (nebo z) režimu podrobnějšího výpisu chybových hlášení.

```
(.)i  
text  
.
```

Příkazem **insert** vkládáme řádky před běžný řádek. Jinak je příkaz totožný s příkazem **a**. Adresa 0 není povolena.

```
(.,.+1)j
```

Příkaz **join** spojí posobě jdoucí řádky do jednoho zrušením znaků nového řádku.

```
(.)kx
```

Příkaz **mark** označí adresovaný řádek malým písmenem *x* (musí být [a-z]). Na tento řádek se později budeme odvolávat adresou '*x*'.

```
(.,.)l
```

Příkaz **list** vypíše adresované řádky následovně: znaky ve výpisu nejednoznačné (tabulátor, backspace) nahradí symbolickým zápisem, netisknutelné znaky se nahradí osmičkovým kódem, dlouhé řádky se rozloží, konec řádku se označí \$. Příkaz **l** smí na řádku následovat za všemi příkazy vyjma **e**, **f**, **r** a **w**.

`(. , .)ma`

Příkaz **move** přesune adresované řádky za řádek s adresou *a*.

`(. , .)n`

Příkaz **number** vypíše adresované řádky tak, že na začátku bude vždy číslo řádku a za ním tabulátor.

`(. , .)p`

Příkaz **print** vypíše adresované řádky.

P

Editor před každým příkazem vypíše **prompt** ve tvaru hvězdičky.

q

Příkazem **quit** se ukončí editor bez ukládání obsahu pracovní kopie do souboru.

Q

Příkazem **Quit** se editor ukončí bez toho, aniž by se kontrolovalo, zda došlo ke změně v pracovní kopii od posledního příkazu w.

$(\$)$ *r soubor*

Příkaz **read** načte obsah souboru *soubor* do editované pracovní kopie.

$(.,.)$ *ta*

Funkce příkazu je podobná příkazu *m*. Rozdíl je v tom, že se provádí kopie, nikoli přesun.



u

Příkaz **undo** odstraní změny v pracovní kopii, které provedl poslední příkaz a, c, d, g, i, j, m, r, s, t, v, G nebo V.

$(1, \$)v/RE/seznam\ příkazů$

Funkce příkazu je podobná příkazu g. Rozdíl je v tom, že se označí řádky, které **nevyhovují** zadanému RE.

```
(1,$)V/RE/
```

Příkaz je podobný interaktivnímu příkazu G. Liší se v tom, že se označí řádky, které nevyhovují zadanému RE.

```
(1,$)w soubor
```

Příkaz **write** zapíše obsah editované pracovní kopie do *souboru*.

$(1, \$)W$  *soubor*

Příkaz **Write** je podobný příkazu `w`. Rozdíl je v tom, že se neukládá do souboru od začátku, ale přidává se nakonec.

$(\$)=$

Příkaz vypíše číslo adresovaného řádku. Běžný řádek se provedením tohoto příkazu nemění.

## *! příkaz shellu*

Řetězec za vykřičníkem se předá k provedení shellu. Předtím se však případný znak procento (%) nahradí zapamatovaným jménem souboru.

## *(.+1)nový řádek*

I pouhá adresa na řádku je příkazem. Běžný řádek se nastaví na tuto adresu a řádek se vypíše. Pouhý stisk klávesy ENTER je totéž, co zadání příkazu `'.+1p'` (užitečné pro procházení souborem).

$(. . .)s/RE/náhrada/$  $(. . .)s/RE/náhrada/g$  $(. . .)s/RE/náhrada/n$ kde  $n$  je číslo 1 až 512

Příkaz **substitute** nahrazuje nalezený vzorek zapsaný RE jeho *náhradou*. Prohledává se každý řádek v zadaném rozsahu adres.

V globální variantě (na konci je uvedeno *g*) se nahradí všechny vyhovující vzorky na řádku; pokud globální indikátor není uveden, potom se nahradí pouze první vyhovující vzorek na každém řádku.

Uvedeme-li číslo  $n$ , potom se nahradí pouze  $n$ -tý výskyt vyhovujícího vzorku na řádku. Pokud se ani na jednom z adresovaných řádků nenajde alespoň jeden vyhovující vzorek, hlásí se chyba.

Na místě oddělovače lomítka (/) se smí uvést libovolný znak vyjma mezery a znaku nového řádku (znak bezprostředně za s určuje, jaký oddělovač bude použit). Běžným řádkem se stane řádek s posledním nalezeným výskytem vzorku.

Znak ampersand (&) nalezený v řetězci *náhrada* se nahradí řetězcem, který vyhovoval RE (na daném řádku, či vyhovujícím vzorku na řádku). Řídicí význam znaku '&' lze v řetězci *náhrada* potlačit uvedením '&'.

V obecnějším významu můžeme v řetězci *náhrada* použít symboly ' $\backslash n$ ', kde  $n$  je číslice. Symbol se nahradí vzorkem, který vyhovoval obsahu  $n$ -té závorky ... uvnitř RE. Závorky se počítají zleva od jedničky a počítá se vždy pořadí otevírací závorky.

Je-li jediným znakem řetězce *náhrada* znak procento (%), potom se pro definici aktuální náhrady použije řetězec *náhrada* z bezprostředně předcházejícího nahrazovacího příkazu. Řídící význam znaku '%' se v řetězci *náhrada* potlačuje tehdy, pokud řetězec obsahuje více než jeden znak, nebo pokud je uvedeno '%'. Zpracováváný řádek lze v řetězci *náhrada* rozdělit na dva (nebo více) zápisem znaku nového řádku, kterému bezprostředně předcházela znak obrácené lomítka (\). Tuto substituci však nelze použít jako součást příkazů *g* nebo *v*.

# Neinteraktivní textový editor sed

```
sed [-n] [-e] příkaz [-f skript] [soubor]
```

-n

Řádky se implicitně nekopírují na výstup.

-e *příkaz*

Příkaz k provedení.

-f *skript*

Soubor se skriptem.

*soubor*

Editovaný soubor.



- Výstup jde vždy na standardní výstup.
- **sed** kopíruje řádek vstupu do **vzorkovacího prostoru**.
- S obsahem vzorkovacího prostoru se provedou všechny příkazy, jejichž adresa vyhovuje.
- Poté se obsah vzorkovacího prostoru zkopíruje na stand. výstup.

## Náhrady:

`[adr1[,adr2]]s/RE/náhrada/příznaky`

- g** Všechny vyhovující řetězce na řádku (impl. první).
- p** Pokud se provedla alespoň jedna náhrada, opíše řádek na výstup.

**w** soubor

Totéž do souboru.

```
A=a:b:c:d
```

```
A=`echo $A|sed 's/:b//'`
```

```
echo $A
```

```
... |sed '1,/něco/d'| ...
```

## Příklad

```
for SOUB in *.txt; do
    JMENO=`echo $SOUB| \
sed 's/\.txt$/.TXT/'`
    mv -f $SOUB $JMENO
done
```

## Příklad

```
JMENO="Ferda Mravenec"  
JMENO=`echo $JMENO| \  
sed 's/\(.*\)□\(.*\)\/\2,\/1/'`
```

## Rozšířené regulární výrazy (egrep, awk)

- Jednoznakový RE následovaný '+' vyhovuje řetězci tvořenému alespoň jedním výskytem JRE (vs. '\*').
- Jednoznakový RE následovaný '?' vyhovuje řetězci tvořenému žádným nebo jedním výskytem JRE.

- Jednoznakovými RE jsou:

`[ :alnum: ]`

písmena, číslice;

`[ :alpha: ]`

písmena;

`[ :cntrl: ]`

řídící znaky (< 32, 127);

`[ :digit: ]`

čísllice;

`[:graph:]`

všechny viditelné znaky (33 až 126);

`[:lower:]`

malá písmena;

`[:print:]`

všechny tištitelné znaky (32 až 126);

`[:punct:]`

viditelné znaky vyjma písmen a číslic;

`[:space:]`

mezera, tabulátory, CR, LF;

`[:upper:]`

velká písmena;

`[:xdigit:]`

čísllice a písmena `[a-fA-F]`.

Symbole používáme uvnitř třídy, např. `[:alnum:]` je totéž co `[0-9A-Za-z]`



- `\w` je synonymem pro `[[:alnum:]]`  
`\W` je synonymem pro `[^[:alnum:]]`
- `\b` je prázdný řetězec na hranici slova  
`\B` je prázdný řetězec "zde není hranice slova"
- JRE následovaný `{m}`, `{m,}`, `{,n}` nebo `{m,n}` je RE, který vyhovuje konkrétnímu počtu výskytů jednoznakového RE.

- RE1|RE2  
Výsledný RE vyhovuje buď RE1 nebo RE2.
- Nejvyšší priorita: ?\*+{  
Střední priorita: konkatenace RE (dva nebo více RE vedle sebe)  
Nejnižší priorita: alternativa |  
Prioritu lze měnit závorkami.

```
ggrep -E 'W{3}' ...
```

```
ggrep -E '(A|W){3}' ...
```

## grep, egrep, fgrep – Global Regular Expression Print

grep

Řádky vyhovující zadanému základnímu RE se vypíší.

grep -E (egrep)

Možnost použití rozšířených RE.

grep -F (fgrep)

Vzorek není RE.

**grep** [-e] *vzorek soubor ...*

**grep** -f *soubor soubor ...*

- i Ignorují se rozdíly mezi malými a velkými písmeny.
- q Potlačí se normální výstup (můžeme testovat návratový kód).
- s Potlačí se chybové výstupy.
- v Invertuje se význam vzorku.
- l Místo vyhovujících řádků se vypisují jména souborů, ve kterých vyhovuje alespoň jeden řádek.

Je-li zadáno více souborů, potom se před každým řádkem vypíše jméno souboru. Chci-li totéž s jedním souborem:

```
grep vzorek soubor /dev/null
```

**GNU grep** v Systému V se často nazývá **ggrep**.

-2 Vypíše 2 řádky před a 2 řádky po vyhovujícím řádku.

# sort

Příkaz **sort** řadí, zatřizuje (merge) a kontroluje seřazení souborů.

-b

Ignoruje se úvodní bílé místo.

-d

Ignorují se všechny znaky vyjma písmen, číslic a bílých míst.

-f

Malá písmena se řadí jako velká.

-n

Řadí se numericky, před číslem se ignoruje libovolně dlouhé bílé místo.

-r

Obrátí se smysl řazení.

-t oddělovač

Uvedený znak se použije pro hledání n-tého pole.

+pos1 [-pos2]

Určení řadicích polí (může být i více).

Pole a znaky se číslují od nuly.

-k pos1,pos2

Totéž, ale čísluje se od jedničky.

## Další významy příkazu **sort**:

-c

Provede se kontrola na seřazenost.

-m

Zatřizuje soubory do jednoho. Zatřizování je rychlejší než řazení. Předpokládá se, že soubory na vstupu jsou již seřazené.

# cut

Ze souboru vybere určitá pole nebo sloupce.

## -b seznam

Určuje se pořadí bajtů na řádku.

## -c seznam

Určuje se pořadí znaků na řádku (od **-b** se může lišit lokalizacemi).

## -f seznam

Určuje se pořadí polí na řádku.



*seznam* např. 1,3,6-8

Čísluje se od jedničky.

-d oddělovač

Oddělovač polí pro -f.

Př: ps -efl|cut -c 7-14,40-53

# uniq

Příkaz kopíruje řádky na stand. výstup s tím, že sousední shodné řádky předá pouze jednou.

Vstup se musí nejprve seřadit příkazem **sort**.

Př: `ps -ef|cut -c1-8|sed 's/ //g'|sort|uniq`

# join

Slučuje soubory dle **slučovacích polí**.

SOUBOR1

SOUBOR2

Novacek Jan

Novacek 1.1.1960

Novak Adam

Novak 31.12.1965

Novotny Petr

Novotny 21.2.1968

```
join SOUBOR1 SOUBOR2
```

Novacek Jan 1.1.1960

Novak Adam 31.12.1965

Novotny Petr 21.2.1968

## -t oddělovač

Implicitně tabulátor.

-1 číslo

-2 číslo

Pořadí slučovacího pole v příslušném souboru (čísluje se od 1). Pouze GNU.

## -o seznam

Popis výstupního řádku. Každý element se zadává ve tvaru  $n.m$  ( $n$  je pořadí souboru,  $m$  je pořadí pole). Seznam musí být zadán jako jedno slovo, položky se oddělují čárkou.

## WC

Vypíše počet bajtů, slov a řádků v souboru nebo souborech.

-c počet bajtů

-w počet slov

-l počet řádků

Př: `ls|wc -l`

# cat

Catenate = sřetěžit

Sřetěží soubory na stand. výstup.

`cat` stnd. vstup → stnd. výstup

`cat soubor` ... soubory → stnd. výstup

`cat -` stnd.vstup → stnd. výstup

Př:

```
cat > soubor
```

```
Text ...
```

```
.....   čtení souboru
```

```
^D       z klávesnice
```

Př:

```
cat soubor      Výpis souboru na obrazovku.
```

## nl – Number Lines

Očísluje řádky.

```
nl
```

```
nl soubor ...
```

```
nl -
```

## od – Octal Dump

Vypíše soubory osmičkově, či jinak.



# head

Vypíše (*implicitně*) 10 prvních řádků.

Př:

`head -5 soubor` Vypíše prvních 5 řádků.

# tail

Vypíše 10 posledních řádků.

Př:

```
tail -2 soubor
```

Vypíše poslední 2 řádky.

```
tail -f logfile
```

Trvale vypisuje.

# pr

Připraví text do podoby pro tisk na tiskárně:

- čísluje stránky
- přidává hlavičky, patičky
- formátuje do sloupců

Př:

*pr -4* upraví do 4 sloupců vertikálně

*pr -a -4* upraví do 4 sloupců horizontálně

*pr -4 -b* vertikálně s vyrovnaným počtem řádků na poslední straně

# tr

Transformuje znaky (*implicitně* ).

Maže znaky.

Ruší opakující se znaky.

## tr

**Transformace znaků:**

**tr** *sada*<sub>1</sub>, *sada*<sub>2</sub> znaky nalezené v sadě<sub>1</sub> se převedou na odpovídající znaky ze sady<sub>2</sub>. Ostatní znaky se ponechají beze změny.

Př:

```
tr a - z A-Z
```

```
tr ' \ 012' @
```

**Rušení znaků:**

**tr -d** ' \000' zruší všechny bajty obsahující binární nulu.

# find - Procházení adresářovým stromem

```
find [cesta...] [výraz]
```

↑ Cesta odkud se adresářový strom prochází.

- Pro každou adresářovou položku provádí výraz
- Výraz se vyhodnocuje zleva doprava a dle priorit
  - ↗ Pravda
- Každý operand výrazu nabývá hodnot
  - ↘ Nepravda

- Výraz se vyhodnocuje jen do okamžiku, kdy je výsledek jasný.  
Např: `a AND b` – `b` se nevyhodnocuje, pokud je `a=0`
- Cesta je implicitně běžný adresář.
- Výraz je implicitně akce **-print**
- Výraz lze složit z těchto operandů:
  - `Voleb` ... Vracejí vždy hodnotu "Pravda"
  - `Testů` ... "Pravda" nebo "Nepravda"
  - `Akcí` ... "Pravda" nebo "Nepravda"
- Neuvedeme-li operátor, použije se implicitní **-and**

# Příklad

`find ~`



Prohledává se  
od domovského  
adresáře

`-name 'core*'`



Test `-name`  
Vrací hodnotu  
"Pravda" při nalezení  
adresář. položky  
jména začínajícího  
core...



Implic.  
`-and`

`-print`



Na stand. výstup  
předá kompletní  
jméno adresářové  
položky a nový řádek  
vždy vrací hodnotu  
"Pravda"

**Pozor: \* je třeba ochránit před expanzí shellu !**



## Volby find:

- depth Nejprve se zpracuje obsah adresáře před adresářem samým.
- follow Budou se procházet symbolické odkazy.

## Testy find:

Numerické hodnoty se vždy zadávají následovně:

- `+n > n`
- `-n < n`
- `n n`

`-atime n` Soubor byl naposledy zpřístupněn před  $n$  dny.

`-empty` Soubor nebo adresář je prázdný.

`-mtime n` Soubor byl naposledy modifikován před  $n$  dny.

- user uživatel
- name jméno
- type
  - b blokový speciální soubor
  - c znakový speciální soubor
  - d adresář
  - p roura
  - f soubor
  - l symbol. odkaz

## akce find:

**-exec** příkaz [argumenty] ;

Provede se zadaný příkaz, je-li návratový kód = 0, vrací se hodnota "Pravda". {} se nahradí aktuální položkou adresáře.

```
find . -type f -exec grep -i smaž {} \; -exec rm {} \;
```

```
find . -type f -exec grep -i smaž {} \; -exec rm {} \; \
```

```
> /dev/null
```

**-ok** příkaz [argumenty] ;

Totéž, příkaz se nechá před provedením potvrdit.

**-print**

Na standardní výstup se předá kompletní jméno souboru od zadané cesty.

Vždy pravda.

# Operátory:

( ... )

Závorkování.

! výraz

Negace výsledku.

výraz<sub>1</sub>            výraz<sub>2</sub>

Logický součin.

výraz<sub>1</sub> -and výraz<sub>2</sub>

Logický součin.

výraz<sub>1</sub> -or výraz<sub>2</sub>

Logický součet.

# cmp

```
cmp soubor1 soubor2
```

Porovnává dva soubory a vypíše offset a obsah prvního rozdílného bajtu.

# diff

Hledá rozdílné a shodné části dvou souborů, či adresářů.  
Rozdíly vypisuje v různých tvarech, implicitně:

**LaR** – Za řádek "L" prvního souboru je přidán rozsah  
řádků "R" ve druhém souboru.

8a 12,15

> .....

> .....

> .....

> .....

**FcT** – řádky "F" prvního souboru jsou nahrazeny řádky "T" druhého souboru.

```
213c214
```

```
< ... První soubor ...
```

```
-- -- --
```

```
> ... Druhý soubor ...
```

**RdL** – řádky "R" z prvního souboru chybí ve druhém souboru za řádkem "L".

```
5,7d3
```

```
< ...
```

```
< ...
```



# patch

Podle výstupu z *diff* a orig. souboru vytvoří soubor nový z porovnávaných souborů.

# tee

Čte stand. vstup a zapisuje na stand. výstup a do souboru.

```
tee[-a] soubor ....  
  ↑  
  append
```

# strings

V souborech hledá sekvence tisknutelných znaků a jenom ty vypisuje.

```
strings [-n číslo] soubor ....
```

↑ *udává minimální délku vypisované  
posloupnosti (impl.4).*

# file

Pokusí se odhadnout typ obsahu zadaného souboru(ů).

## **file soubor ...**

*Příkaz v souboru hledá řetězce podle definice v souboru /etc/magic*

# du

Vypisuje informace o obsazení diskové kapacity.

```
du [-volby] cesta ...
```

Kapacitu souboru/ adresáře vypisuje v blocích (512 B) nebo KB (viz. man du ).

- a Všechny soubory, ne jenom adresáře.
- k V KB.
- s Pro každý zadaný argument vypíše celkovou kapacitu.

# tar – Tape ARchiver

- Vytváření / Rozbalování / Prohlížení archivačních souborů.
- Určeno pro archivaci adresářových stromů.

```
tar [ -volby ] jméno ....  
  ↑minus není povinné
```

## Volby:

- c** – vytvoření archivu
- x** – rozbalení archivu
- t** – výpis adresáře z archivu

- Př: `tar c *`  
*Všechny soubory běžného adresáře vč. podadresářů uloží do archiv. souboru a zapíše na /dev/rmt0 .*
- Př: `tar x`  
*Všechny soubory a podadresáře z arch. souboru čteného z /dev/rmt0 rozbolí na místo podle jména v archivu:*
  - *absolutní /../..*
  - *relativní ./../..*
- Př: `tar x soubor`  
*Z arch. souboru rozbolí pouze uvedený soubor (nebo adresář vč. podadresářů).*

# Volby tar(u):

## f

Následující argument se použije jako jméno archivačního souboru místo /dev/rmt0

Je-li jméno souboru '-', čte/zapisuje se archiv z/do standard. vstupu/výstupu.

Př:

```
tar cf adresář.tar adresář
```

```
tar tf adresář.tar | more
```

```
adresář/
```

```
adresář/ soubor1
```

...

```
tar xf adresář.tar adresář/soubor1
```

Př: Využití tar na kopírování celé adresářové struktury.

```
cd odkud; tar cf - . |(cd kam; tar xf - )
```



Normálně tar nic nevypisuje.

Např.: `cv` vypisuje kompletní jména souborů vkládaná do archivu.

Např.: `tv` vypisuje o souboru více informací.

Informace vypisuje na stand. výstup, nebo na std. chybový výstup při `cf - .`

## Jiný archivační program: `cpio`

# compress

Komprimuje uvedené soubory.

**compress soubor ...**

soubor → soubor.Z

# uncompress

**uncompress soubor.Z ...**

soubor.Z → soubor

# zcat

## zcat soubor.Z ...

Vypisuje rozbalený obsah komprimovaných souborů.

`gzip` soubor → soubor.gz

`gunzip` soubor.gz → soubor

`zcat` soubor.gz

*gzip, gunzip, zcat dosahují lepší kompresní poměr.*

*Nejsou součástí běžných UNIXů.*

# zip

```
zip soubor.zip soubor
```

**unzip**  
**zipsplit**

```
zipsplit [-n velikost] soubor.zip
```

Rozdělí *zip* archiv na více archivů velikosti maximálně *velikost*.

## dd - Konvertuje a kopíruje soubor

Kopíruje stnd. vstup na stnd. výstup.

```
dd [operandy ... ]
```

`bs = n` Nastaví se velikost vstupního a výstupního bloku.

`ibs = n` Vstupní blok.

`obs = n` Výstupní blok.

`if = soubor` Vstupní soubor místo stnd. vstupu.

`of = soubor` Výstupní soubor místo stnd výstupu.

Př: Úprava velikosti bloku při zápisu na pásku (podle dokumentace zařízení):

```
tar cvf - *|dd bs=60k of=/dev/rst12
```

Další operandy viz *man dd* .

# who

**who**

**who am i**

**w**

Kdo je přihlášen a co dělá.

# last

## **last [uživatel]**

Kdo, kdy, odkud a jak dlouho byl přihlášen.



# finger

```
finger @počítač
```

Kdo je přihlášen na vzdál. počítači.

```
finger uživatel@počítač
```

Informace o uživateli.

# write

**write** uživatel [terminál]

Zpráva lokálnímu uživateli, případně i na konkrétní terminál, je-li přihlášen vícekrát.

# mesg

```
mesg -n
```

Zakáže se příjem zpráv zasílaných příkazem *write* .

```
mesg -y
```

Povolí.