Úvod do operačního systému GNU / Linux
Přednášky

Jan Pytel

<jan.pytel@fsv.cvut.cz>

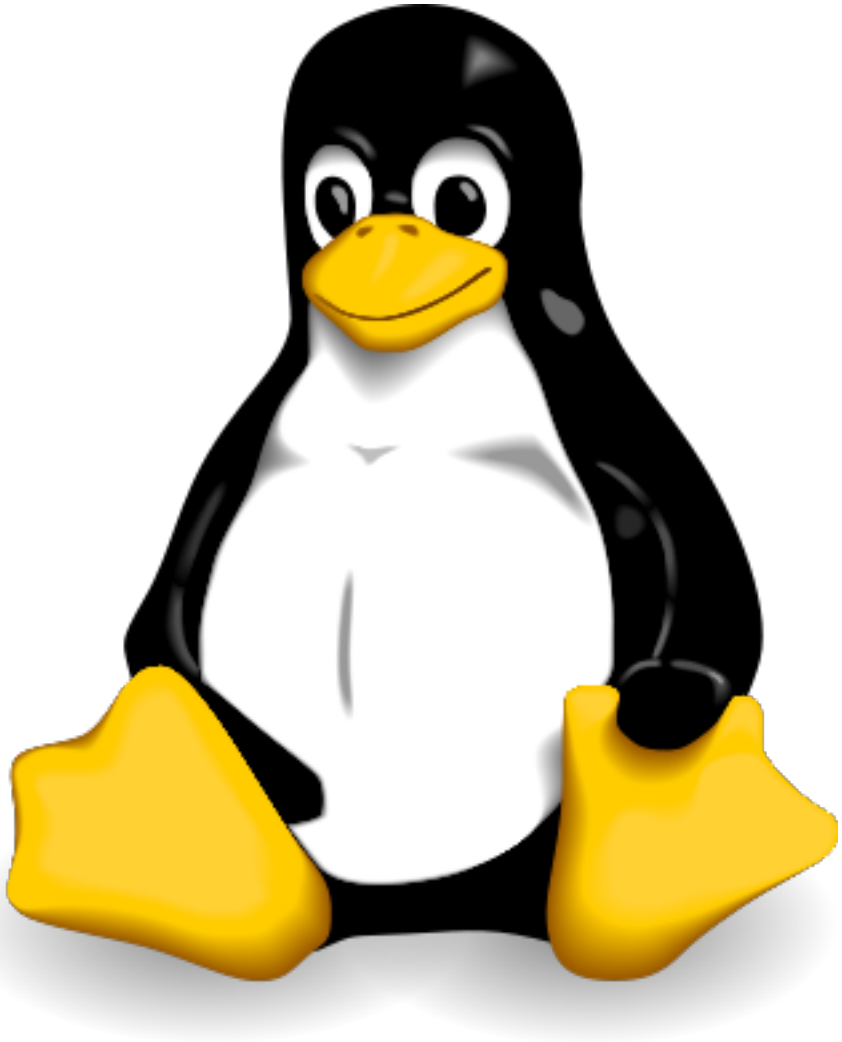
České vysoké učení technické v Praze
Fakulta stavební
Katedra mapovaní a kartografie

Přednášky ZS 2010/2011

Operační systém GNU / Linux

Operační systém GNU / Linux

Operační systém GNU / Linux je tvořen jádrem (*Linux) a aplikačním softwarem (GNU)*



Základní informace o GNU projektu

- založení *GNU projektu — významný okamžik v historii svobodného software*
- *GNU's Not Unix*
- *projekt vnesl mezi programátory "ducha spolupráce"*
- *založen roku 1984 Richardem Stallmanem*
- *hlavní cíl GNU projektu — snaha vytvořit kvalitní volně šiřitelný software*

Základní informace o GNU projektu

- vytvořit programy, které budou zadarmo a nebudou nikoho omezovat v jejich používání (každý může program vylepšit, studovat, modifikovat, nebo použít část ve svém programu)
- http://www.gnu.org
- za svobodný software jsou považovány programy poskytující uživatelům následující čtyři svobody:
  - svoboda spustit program za libovolným účelem
  - svoboda přístupu ke zdrojovému kódu
  - svoboda redistribuce kopií
  - svoboda vylepšování/modifikace programu

Richard Stallman

- narozen roku 1953
- programátor v *MIT AI Laboratory*
- *zakladatel zakladatel hnutí svobodného software, GNU projektu, Free Software Foundation*
- *aktivně se podílel na tvorbě mnoha programů:*
  GNU Emacs,
  GNU C Compiler,
  GNU C Debugger

General Public Licence — GPL

- bylo nutné zajistit, aby nedošlo k pozdějšímu “zneužití” celého projektu, nebo jeho částí
- licence v kompletním znění
  http://www.gnu.org/copyleft/gpl.html
- licence obsahuje řadu ustanovení a podmínek pro kopírování, distribuci a modifikaci GNU programů
- *Nejdůležitější ustanovení: Vše co bylo vytvořeno z programů distribuovaných pod GNU licencí tuto licenci automaticky přebírá*

Linux

- jádro operačního systému (zajišťuje spolupráci s periferiemi)
- 1991 - student Linus Torvalds
- ideově vychází z operačního systému UNIX
- víceuživatelský
- multitasking

Linus Torvalds

- narozen roku 1969
- student *University of Helsinki*
- *inspirován MINIXEM*
- *roku* 1991 *začíná práci na LINUXu*

Linux - dopis Torvaldse

From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki
Hello everybody out there using minix - I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things). I've currently ported bash(1.08) and gcc(1.40), and things seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome, but I won't promise I'll implement them :-) Linus (torvalds@kruuna.helsinki.fi) PS. Yes - it's free of any minix code, and it has a multi-threaded fs. It is NOT protable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-(.

GNU / Linux — Distribuce

- Distribuce: jádro + řada aplikací
- “Klasické distribuce”
  - *Debian* — *www.debian.org*
  - *Gentoo* — *www.gentoo.org*
  - *Ubuntu* — *www.ubuntu.com*
  - *Red Hat* — *www.redhat.com*
- “Live distribuce”
  - *Knoppix* — *www.knoppix.org*
  - *Danix* — *www.danix.cz*

Začínáme

Přihlášení do systému

Po zavedení systému se uživatelé mohou přihlašovat do systému (pokud pro ně existuje uživatelský účet). Pro přihlášení lze využít textovou konzoli, či grafickou nástavbu. Každý uživatel musí znát:

uživatelské jméno jméno pod kterým zná uživatele systém

uživatelské heslo netriviální heslo

Po zadání uživatelského jména se zobrazí informace o vložení hesla, při stisku kláves se na konzoli nic nezobrazuje.

```
server login: novak
Password:
```

Přihlášení ke vzdálenému počítači

OpenSSH SSH Client - `ssh` - program pro přihlášení ke vzdálenému počítači (počítač u kterého fyzicky nesedíme). Po přihlášení lze na vzdáleném počítači provádět příkazy.

```
ssh jmeno_serveru -luzivatelske_jmeno
```

jmeno_server jméno serveru ke kterému se uživatel přihlašuje

uzivatelske_jmeno určuje uživatele logujícího se na server

Ukázka

```
ssh josef.fsv.cvut.cz -lpytel
```

Přihlášení ke vzdálenému počítači — ukázka

```
student@b870:~$ ssh gama.fsv.cvut.cz -lpytel
pytel@gama's password:
Linux gama 2.6.17-2-686 #1 SMP Wed Sep 13 16:34:10 UTC 2006 i686

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.

Last login: Sun Oct  1 22:51:23 2006 from k153pytel.fsv.cvut.cz
pytel@gama$
```

Neúspěšné přihlášení:

```
honza@prasatko:~$ ssh gama.fsv.cvut.cz -lpytel
Password:
Password:
Password:
Permission denied (publickey,password).
honza@prasatko:~$
```

Prompt

Po přihlášení do systému se uživateli zobrazí řádek začínající sekvencí znaků (nazývá se *prompt*), prompt obvykle končí znakem $. Znakem $ signalizuje uživateli, že je připraven na příkazy z klávesnice.

```
uzivatel@server:aktualni_adresar$
```

uzivatel	jméno serveru ke kterému se uživatel přihlašuje
server	určuje uživatele logujícího se na server
aktualni_adresar	adresář ve kterém se uživatel právě nachází

Ukázka:

```
pytel@josef:/var/lib$
```

Zkratky pro práci v terminálu

Terminál během práce nabízí uživateli celou řadu klávesových zkratek a aplikací pro usnadnění editace, pohybu v historii, vyhledávání, ...

Ctrl + Alt + Fn - přepnutí do n–tého virtuálního terminálu, defaultní počet je 6.

Ctrl + Alt + F7 - přepnutí do GUI - X Windows System

Ctrl + a - posunutí kurzoru na začátek řádku

Ctrl + e - posunutí kurzoru na konec řádku

Alt + b - posunutí kurzoru na předchozí slovo

Alt + f - posunutí kurzoru na další slovo

Ctrl + u - smazání řádky

Tab - automatické doplňování

Zjištění uživatelského jména, vypsání přihlášených uživatelů

Pro vypsání uživatelského jména právě přihlášeného uživatele, příkaz `whoami`. Pro vypsání všech, aktuálně přihlášených, uživatelů na serveru použijeme příkazy `w` či `who`.

Ukázka:

```
pytel@gama:~$whoami
pytel
pytel@gama:~$
```

Souborový systém

- *filesystem* - způsob organizace informací (počítačových souborů) na médiu (disk, CD, $\cdots$)
- Linux obsahuje řadu FS: ext2, etx3, ReiserFS, JFS, ISO 9960
- / - *rootovský adresář*, obsahuje všechny soubory a adresáře
- příklady některých adresářů v *rootovském adresáři*

  bin – spustitelné soubory
  etc – obsahuje konfigurační soubory
  home – obsahuje domovské adresáře uživatelů

Pohyb mezi adresáři – seznam příkazů

Seznam příkazů

- pwd výpis aktuálního adresáře, kde se uživatel nachází
- cd změna aktuálního adresáře
- ls výpis obsahu aktuálního adresáře

Příkaz `pwd` vypíše aktuální (pracovní) adresář:

```
pytel@gama:~$pwd
/home/pytel
```

Příkaz cd

Příkaz `cd` (akronym od *change directory*) mění aktuální adresář.
Syntaxe

```
cd jmeno_adresare
```

`jmeno_adresare` může být adresováno *relativně* či *absolutně*.

Znaky se speciálním významem:

- `/` rootovský adresář
- `~` domovský adresář
- `.` aktuální (pracovní) adresář
- `..` nadřazený adresář

Absolutní adresování

Příkaz `cd` bez argumentu přechází do domovského adresáře.

```
pytel@gama:/var/lib$cd
pytel@gama:~$
```

Absolutní adresování: adresa vždy začíná znakem / za nímž následuje daný adresář (adresáře). Jednotlivé adresáře jsou odděleny znakem /. Např.

```
/var/lib/cvs/CVSROOT
```

znamená: v *rootovském adresáři* existuje adresář `var`, tento adresář obsahuje adresář `lib` ...

Relativní adresování

Aktuální adresář je adresář v kterém se právě uživatel nachází. Pokud při zadání adresáře (či souboru) nezačíná cesta znakem `/`, jedná se o relativní adresování a cesta se vztahuje k aktuálnímu adresáři.

Příklady:

adresar	označuje adresář `adresar` který se nachází v aktuálním adresáři
./adresar	stejné jako předcházející
../adresar	označuje adresář `adresar` který se nachází v rodičovském adresáři
adresar/adresar1	označuje adresář `adresar1` který se nachází v podadresáři `adresar`

Jména souborů

Každý soubor a adresář má alespoň jedno jméno v *file systému*. Může být tvořen “libovolným počtem znaků” a nesmí obsahovat znak `/`.
Nedoporučené znaky ve jménech souborů:
`?, *, [, ], \, (, ), {, }, |.`

Soubor jehož název začíná znakem `.` je soubor skrytý a defaultně se nezobrazuje při výpisech adresáře programem `ls`.

.bashrc	skrytý soubor
pokus.txt	soubor s koncovkou `txt`, zřejmě textový soubor

Operační systém GNU / Linux neurčuje, zda je soubor spustitelný na základě přípon (OS Windows — `.bat`, `.exe`, `.com`).

Příkaz `ls`

Příkaz **ls** - příkaz vypisuje obsah aktuálního adresáře. Vypíše (abecedně seřazená) jména všech souborů a adresářů.

```
ls [parametry] [soubor]
```

parametry nastavení kterými uživatel specifikuje, jak bude výpis vypadat a co bude obsahovat

soubor informace o daném souboru či adresáři

Ukázka:

```
pytel@gama:~/DS/2006$ls
CVUT-Geoinformatics  FIG  Mnichov  GISOStrava
```

Příkaz `ls` — parametry

Příkaz **ls** můžeme spustit s řadou parametrů. Seznam jednotlivých parametrů následuje:

- `-l` výpis bude ve speciálním “long” formátu
- `-a` ve výpisu budou i soubory začínající znakem `.`
- `-R` rekursivní vylistování adresářů
- `-F` za každý vypsaný adresář či soubor je přidán identifikátor (`{ * / = > @ |`)

- `ls -l /var/` výpis adresáře `/var` ve formátu “long”
- `ls -a /` výpis kořenového adresáře, vypsány i skryté soubory
- `ls -la` výpis aktuálního adresáře ve formátu long, vypsány i skryté soubory.

Příkaz `ls -l`

```
$ls -l
drwxrwsr-x 2 root src 4096 Mar 18  2006 Emptydir
-r--r--r-- 1 root src  495 Mar 18  2006 checkoutlist
```

drwxrwsr-x	práva přístupu k souboru
2	kolikrát je soubor odkazován
root	vlastník souboru
src	skupina která soubor vlastní
4096	velikost souboru
Mar 18 2006	čas poslední modifikace
Emptydir	název adresáře (či souboru)

Filesystem Hierarchy Standard

Filesystem Hierarchy Standard definuje strukturu adresářů v operačních systémech typu GNU/Linux. Dále definuje, jaké soubory/programy musí konkrétní adresáře obsahovat, např. adresář **/bin**.

FHS je de facto standardem, který by měli dodržovat lidé přispívající či udržující GNU/Linux distribuce.

Přesné znění standardu je umístěno na adrese:

http://www.pathname.com/fhs/

Příkazy `man` a `info`

Nápověda — příkaz `man`

Příkaz man vypisuje na standardní výstup informace o programu, jehož název byl zadán jako parametr.

```
man program
```

Man — manuálové stránky jsou dělené do 6 sekcí, dle tématu.

Klávesy pro ovládání:

- Space pohyb dolů
- PgUp/PgDown pohyb v nápovědě
- q klávesa ukončí příkaz `man`
- / klávesa otevře režim vyhledávání
- n v režimu vyhledávání posune na další výskyt hledaného řetězce.

Nápověda — příkaz `info`

Příkaz info je další možností jak zobrazit nápovědu v systému GNU / Linux. Nápověda je řazena do jednotlivých sekcí. Každá sekce (kapitola) je na začátku řádku označena znakem `*`.

```
info program
```

Klávesy pro ovládání:

- u — pohyb v hierarchii směrem nahoru
- PgUp/PgDown — pohyb v nápovědě
- q — klávesa ukončí příkaz `info`
- / — klávesa otevře režim vyhledávání
- n — v režimu vyhledávání posune na další výskyt hledaného řetězce.

Nápověda pro vestavěné příkazy — `help`

Vestavěné příkazy (anglicky builtin commands) jsou příkazy, které jsou obsaženy přímo v shellu (více dále). Tyto vestavěné příkazy, například `cd`, nemají obvykle vytvořené manuálové či info stránky. Pokud hodláme získat informaci k těmto příkazům, je nutno použít vestavěný příkaz `help`.
Nápověda k příkazu `cd` má následující tvar:

```
help cd
cd: cd [-L|-P] [dir]
    Change the current directory to DIR.  The variable $HOME is the
    default DIR.  The variable CDPATH defines the search path for
    the directory containing DIR.  Alternative directory names in CDPATH
    are separated by a colon (:).  A null directory name is the same as
    the current directory, i.e. `.'.  If DIR begins with a slash (/),
    then CDPATH is not used.  If the directory is not found, and the
    shell option `cdable_vars' is set, then try the word as a variable
    name.  If that variable has a value, then cd to the value of that
    variable.  The -P option says to use the physical directory structure
    instead of following symbolic links; the -L option forces symbolic
    links to be followed.
```

Práce s adresáři a soubory

Vytváření adresářů

Adresář se vytváří příkazem `mkdir` s parametrem jména adresáře.

```
mkdir jmeno_adresare
```

V jménu se uplatní relativní a absolutní adresování, adresář nesmí existovat a uživatel musí příslušná práva.

Ukázka

```
$mkdir auto
$mkdir auto/car
$mkdir auto/car/xyz/abc
mkdir: cannot create  directory 'auto/car/xyz/abc': No
directory
```

Mazání adresářů

Příkaz `rmdir` zruší prázdný adresář.

```
rmdir jmeno_adresare
```

Nelze rušit adresáře které nejsou prázdné a adresáře pro jejichž zrušení nemá uživatel příslušná práva.

Ukázky:

```
$rmdir auto/car
$rmdir auto
$rmdir auto/car/xyz/abc
rmdir: 'auto/car/xyz/abc': No such file or directory
```

Vypsání souboru

Obsah souboru na standardní výstup lze vypsat pomocí příkazů `cat`, `less` a `more`.

```
cat soubor
less soubor
more soubor
```

Příkaz `cat` soubor pouze vypíše, příkazy `more` a `less` výpis "stránkují". Příkazy `more` a `less` umožňují dále vyhledávání zadaného vzoru pomocí klávesy.

u	pohyb v hierarchii směrem nahoru
PgUp/PgDown	pohyb ve výpisu, pouze `less`
q	klávesa pro ukončení příkazu
/	klávesa otevře režim vyhledávání (`less` a `more`)

Vytvoření souboru — příkaz `touch`

Příkaz `touch` mění timestamp pro soubor zadaný parametrem.

```
touch soubor
```

V případě že soubor neexistuje, je vytvořen prázdný soubor.

```
$ls -l
-rw-r--r--  1 honza honza 0 2006-10-23 11:37 soubor
$touch soubor
$touch soubor1
$ls -l
-rw-r--r--  1 honza honza 0 2006-10-23 11:38 soubor
-rw-r--r--  1 honza honza 0 2006-10-23 11:38 soubor1
```

Přejmenování a přesun souboru/adresáře — příkaz `mv`

Příkaz `mv` přesouvá soubor/adresář do jiného adresáře, zároveň slouží k přejmenování souborů.

Syntaxe:

```
mv soubor cil
```

Pokud **cil** je adresář, dojde k přesunu souboru **soubor** do tohoto adresáře. Jestliže **cil** je souborem, dojde k přejmenování a přesunutí souboru **soubor** do souboru **cil** (pokud soubor **cil** existuje, je jeho obsah nahrazen obsahem souboru **soubor**).

```
~$mkdir ukazka
~$mv /etc/passwd ~/ukazka/
~$mv /etc/passwd ~/ukazka/passwd-zaloha
```

Kopírování souboru — příkaz `cp`

Příkaz `cp` kopíruje soubory do zadaného cíle. Jestliže je cílem adresář, lze kopírovat více souborů najednou.

Syntaxe:

```
cp soubor cil
```

Pokud **cil** je adresář, dojde ke kopírování souboru **soubor** do tohoto adresáře. Pokud **cil** je souborem, dojde k nahrazení obsahu tohoto souboru, obsahem souboru **soubor**. Jestliže **cil** neexistuje, příkaz předpokládá, že se jedná o neexistující soubor kam se má soubor zkopírovat. Při použití rekurzivního parametru `-r` je možno kopírovat rekurzivně celé adresáře.

```
~$mkdir ukazka
~$cp /etc/passwd ~/ukazka/
~$cp /etc/passwd ~/ukazka/passwd-zaloha
```

Rušení souboru — příkaz `rm`

Příkaz `rm` maže (ruší) soubory.

Syntaxe:

```
rm soubory
```

Příkaz `rm` umí i rušit neprázdné adresáře, pokud jako parametr uvedeme parametr **-r**:

```
rm -r neprazdny_adresar
```

Upozornění: Filesystémy pro OS GNU / Linux nemají obvykle funkci *undelete*.

Secure copy — příkaz `scp`

Secure copy - `scp` - program pro kopírování souborů mezi vzdálenými počítači (obvykle z lokálního počítače do vzdáleného počítače a naopak).

```
scp zdroj cil
```

Argumenty programu `scp` **zdroj** a **cil** mají následující syntaxi **uzivatel@server:soubor**, kde:

uzivatel	specifikuje pod kterým uživatelem bude kopírování na daném serveru probíhat
server	jméno serveru
soubor	cesta k souboru (či soubor). Pokud je prázdná, předpokládá se domovský adresář

Ukázky: Secure copy — příkaz `scp`

Ukázky:

```
scp /etc/passwd pytel@gama.fsv.cvut.cz:

scp /etc/passwd pytel@gama.fsv.cvut.cz:/tmp/xxxx

scp pytel@gama.fsv.cvut.cz:/etc/passwd .

scp pytel@gama.fsv.cvut.cz:/etc/passwd \
    pytel@josef.fsv.cvut.cz:passwd-gama
```

Změna přístupových práv k souboru

Přístupová práva k souboru se nastavují pro “tři skupiny uživatelů” (viz příkaz `ls -l`):

- vlastník
- skupina
- ostatní uživatele

Každá skupina uživatelů má trojici přístupových práv, která lze libovolně skládat dohromady:

- r obsah souboru lze číst (např. `cat, less, ...`)
- x lze měnit obsah souboru
- x soubor je spustitelný

Pokud některé právo není nastaveno, zobrazuje se v příkazu `ls` na příslušném místě znak **-**.

Změna přístupových práv k souboru — příkaz `chmod`

Příkaz **chmod** mění přístupová práva pro daný soubor či adresář.

```
chmod parametry soubor
```

Parametry označují jaké práva pro kterou skupinu budou nastaveny (pro nastavení práv souboru musím mít daný uživatel “oprávnění”) a mají následující tvar:

```
skupinyOPERATORprava
```

Příkaz **chmod** mění pouze práva pro jednotlivé skupiny uživatelů, změna vlastníka či skupiny se provádí příkazem `chown`.

Změna přístupových práv k souboru — příkaz `chmod`

Skupiny:

- u vlastník souboru
- g skupina uživatelů
- o ostatní
- a všichni

Práva:

- r obsah souboru lze číst (např. `cat, less, ...`)
- x lze měnit obsah souboru
- x soubor je spustitelný

Změna přístupových práv k souboru — příkaz `chmod`

Operátor:

- `=` nastav právo
- `+` přidej dané právo/práva
- `-` odeber dané právo

Ukázky:

```
$chmod o-x    soubor
$chmod u+rw   soubor
$chmod go=r   soubor
$chmod ugo=   soubor
$chmod u=rwx,o=r,g= soubor
```

Symbolické odkazy — příkaz `ln`

Symbolický odkaz je soubor, který ukazuje na jiný soubor (vzdálená analogie se zástupcem z OS Windows). Symbolické odkazy mohou odkazovat jak na soubor tak adresář. V případě, že použijeme příkaz který čte obsah symbolického linku, pracujeme vlastně se souborem, na který symbolický link odkazuje. Symbolické linky se vytváří programem `ln` s parametrem `-s`.

```
ln -s soubor symbolicky_odkaz
```

```
~$ln -s /etc/passwd passwd
~$ls -l passwd
lrwxrwxrwx 1 pytel pytel 11 Oct 30 08:21
                         passwd -> /etc/passwd
```

Práce s obsahem souborů

Formátování obsahu souboru — `fmt`, `fold`

Příkazy **`fmt`** a **`fold`** formátují obsah souboru. Příkaz **`fmt`** čte data ze vstupu a na standardní výstup vypisuje zformátované odstavce dle zadané šířky řádku (standardně 75 znaků) . Šířku lze měnit parametrem **`-w šířka`**.

Např.:

```
$fmt -w 80 soubor
```

Příkaz **`fold`** odstavce ze vstupu neformátuje, pouze zalamuje řádky delší než je nastavena maximální délka řádku (defaultně 80 znaků).

Např.:

```
$fold -w 80 soubor
```

Rozdělování souboru — `split`

Příkaz **`split`** rozděluje soubor (či data ze standardního vstupu) do více souborů dle zadaného kritéria. Vytvořené soubory mají názvy **`prefixaa`**, **`prefixab`**, ...

```
$split [parametry] [soubor [prefix]]
```

Parametry:

- `-a N` použije suffix délky *N*
- `-b N` počet *N* bytů do jednotlivého souboru (za číselnou hodnotu lze připojit jeden ze znaků (b, k, m)
- `-l N` počet řádek do každého souboru

Příklad rozdělení souborů na soubory o velikosti 2 kB:

```
$split -b 2k soubor soubor-cast-
```

“Řádkové” spojování souborů — `join` a `parse`

Příkaz **`paste`** spojuje (dva a více souborů): řádky na odpovídajících si pozicích z jednotlivých souborů zapisuje na standardní výstup jako řádek jeden, jednotlivé řádky souborů odděluje tabulátorem (lze změnit parametrem **`-d znak`**.

```
$paste soubor1 soubor2
```

Příkaz **`join`** spojuje dva a více souborů řádkově, jestliže mají identické “spojovací pole”, výsledek je poslán na standardní výstup. Spojovací pole je defaultně první pole (pole jsou oddělena bílými znaky). *Soubory musí být seřazeny dle spojovacího pole!*

```
$join [parametry] soubor1 soubor2
```

Porovnávání souborů — příkaz `diff`

Příkaz **`diff`** porovnává soubory řádek po řádku. V případě že některé řádky se liší, vypíše rozdílné řádky na standardní výstup a vypíše pozice řádků v jednotlivých souborech:

```
$diff soubor1 soubor2
```

Parametry:

- `-c N` výpis N (defaultně tři) okolních řádek
- `-i` nerozlišuje mezi velkými a malými písmeny
- `-b` ignoruje změny v počtu “bílých znaků”

Výstup má následující tvar:

```
změna
< řádky prvního souboru
---
> řádky druhého souboru
```

Porovnávání souborů — příkaz `diff`

Výstup programu **`diff`** obsahuje vždy položku změna která má následující formát: `cislo1ZMENAcislo2`, kde

- cislo1 číslo příslušného řádku v prvním souboru
- cislo2 číslo příslušného řádku v druhém souboru
- ZMENA typ změny:
  - a v prvním souboru se řádka nevyskytuje
  - c v prvním a druhém souboru jsou řádky různé
  - d v druhém souboru daná řádka neexistuje

Ukázka:

```
$diff Pit_Pendulum Pit_Pendulum2
657c657
< Something unusual -- some change which
---
> Something unusual --- some change which
```

Vytváření MD5 kontrolních součtů — `md5sum`

Příkaz **`md5sum`** počítá MD5 kontrolní součet pro dané soubory. Dnešní moderní GNU/Linux distribuce již standardně začínají mít i další programy pro výpočet kontrolního součtu, např. `sha1sum`.

```
$md5sum soubory
```

Ukázka:

```
$md5sum * > md5sums-$(date -I)
$cat md5sums-2006-11-19
7ff455de3f8e76dfdc5f76e7dd1600d4  Makefile
0ebe8d7d8618944d55d09b4f9d828631  osyl.aux
37b2df7c2e71cc194c5610828c42b1d5  osyl.nav
3806243f927b6a0f7f1ab73ab6c0a61c  osyl.pdf
2a3404f8f1f63d2d81813f4fce525bb5  osyl.tex
48f8b1e635224f98cebcca55d88ee526  osyl.toc
3bed3b461f5332ab8164e706ff701481  osyl.vrb
```

Změny obsahu souboru — `tr`

Příkaz **tr** načítá data ze vstupu a na těchto datech provádí transformace (pracuje se znaky). Příkaz **tr** poskytuje následující operace:

- měnit znaky
- redukovat opakující se znaky (vtěsňovat)
- mazat znaky
- mazat znaky a následně redukovat opakující se znaky

```
$tr [OPTION]... SET1 [SET2]
```

SET1 - obsahuje znaky které se budou nahrazovat, nebo mazat
SET2 - obsahuje nové znaky

Množiny znaků mohou obsahovat i seznam znaků: [:alpha:], [:digit:], [:lower:], [:upper:], atd.

```
cat /etc/motd | tr abc xyz
cat /etc/motd | tr '[:lower:]' '[:upper:]'
cat /etc/motd | tr -d a
```

Vyhledávání v souborech — regulární výrazy

Vyhledávání v souborech — příkaz `grep`

Příkaz **grep** prochází textovým souborem *řádku po řádce* a hledá zadaný vzor. V případě, že vzor nalezne, příslušnou řádku vypíše na standardní výstup:

```
grep parametry vzor soubor
```

Vzorem může být řetězec znaků, některé znaky mají speciální význam (často je vhodné obalit vzor dvojitými uvozovkami: “vzor”).
Často používané parametry:

- `-i` nerozlišuje se mezi velkými a malými písmeny
- `-v` vypíše pouze řádky které neobsahují zadaný vzor
- `-l` místo řádek jsou vypsány pouze soubory dané řádky obsahující
- `-E` interpretuje vzor jako “extended regular expression”

Příkaz `grep` – regulární výrazy

Vzor pro příkaz `grep` může obsahovat znaky se speciálním významem, takový vzor se nazývá *regulární výraz*. Regulární příkazy se pro řadu GNU příkazů dělí na:

- základní regulární výrazy (“basic regular expressions”)
- rozšířené regulární výrazy (“extended regular expressions”).

Pro příkaz `grep` platí, že oba typy regulárních výrazů se dělí pouze syntaxí: v základních regulárních výrazech znaky se speciálním významem ?, +, {, |, (a) ztrácí svůj význam a musí být použity se znakem \: \?, \+, \{, \|, \(a \).

Příklad:

```
$ echo "axxxxy" | grep -E "ax{4}y"
axxxxy
$ echo "axxxxy" | grep  "ax\{4\}y"
axxxxy
```

Příkaz `grep` – regulární výrazy

Znaky se speciálním významem:

^	začátek řádky
$	konec řádky
\< \>	ohraničení slova
(slovo1|slovo2)	logické nebo
(sekvence)	závorky označují sekvenci která se může opakovat
.	jeden libovolný znak
[abc]	libovolný znak ze znaků `abc`
[^abc]	žádný ze znaků `abc`
\	potlačí význam metaznaků, např. *

Příklady:

```
"a?"
"^[pv]es"
"2[1-7][1-5]0$"
"abcdefg"
```

Příkaz `grep` – regulární výrazy

Znaky se speciálním významem (opakování):

- `?` předchozí znak se může vyskytovat **maximálně** jednou
- `*` znak který znaku * předchází, může být uveden (0, . . .)
- `+` předchozí znak se může vyskytnout **minimálně jednou**
- `{n}` předchozí znak se může vyskytnout právě **n**krát
- `{n,}` předchozí znak se může vyskytovat **nejméně n**krát
- `{n,m}` předchozí znak se může vyskytovat **nejméně n**krát a maximálně **m**krát

Příklady:

```
echo "aaXYZbbbc" | grep -E "a{1,4}.*b{3}c?"
echo "aacXjarojaro" | egrep "[abc]+.*(jaro|leto){2}$"
echo "Je krasny den" | grep -E "\<krasny\>"
echo "krk" | grep -E "(.).\1"
```

Příkaz `grep` – regulární výrazy

Znaky se speciálním významem (opakování):

- [:alnum:][0-9A-Za-z]
- [:alpha:][A-Za-z]
- [:digit:][0-9]
- [:lower:][a-z]
- [:space:]
- [:upper:][A-Z]

Příklady:

```
"[[:digit:]]{2}[0156][[:digit:]][0123] \
 [[:digit:]]/[[:digit:]]{4}"
```

Vypsání prvních `n` řádek souboru

Příkaz **head**, resp. **tail** vypíše prvních, resp. posledních **n** řádek souboru, defaultně vypisuje 10 řádek.

```
head -n pocetradek soubor
tail -n pocetradek soubor
```

Jestliže soubor obsahuje méně řádek, než udává parametr **pocetradek**, je vypsáno právě tolik řádků, kolik jich existuje v daném souboru.

Číslování řádek souboru, informace o souboru

Příkaz **nl**, resp. **cat -n** očísluje všechny řádky souboru a očíslované řádky vypíše na standardní výstup (tedy nemodifikuje daný soubor).

```
nl soubor
cat -n soubor
```

Příkaz **wc** spočítá pro daný soubor všechny řádky, slova a znaky. Parametry `-l`, `-w`, `-m` vypíší pouze počet řádků, slov, znaků.

```
wc  soubor
wc -l soubor
```

Standardní vstup a výstup

Standardní vstup, výstup — přesměrování

Většina příkazů systému GNU / Linux čte data defaultně ze standardního vstupu a vypisuje je na standardní výstup, chyby vypisuje do chybového výstupu. Implicitně je standardním vstupem a výstupem terminál.
Přesměrovat standardní vstup a výstup lze pomocí znaků $>$ a $<$.

```
$cat /etc/passwd > soubor
$cat < soubor > soubor1
```

Jestliže soubor do kterého přesměrováváme existuje, je jeho obsah přepsán.

Standardní vstup, výstup — přesměrování II

Lze přesměrovat standardní výstup pomocí znaků », pokud nyní existuje soubor do kterého přesměrováváme, není obsah souboru přepsán a zapisuje se na konec souboru.

```
$echo pokus  > soubor
$echo pokus1 > soubor
$echo pokus2 >> soubor
$
$cat soubor
pokus1
pokus2
$
```

Chybový výstup — přesměrování

Programy vypisují chyby do chybového výstupu (standardního chybového výstupu). Defaultně je standardním chybovým výstupem terminál, je však možno chybový výstup přesměrovat do souboru pomocí **2>**:

```
$program 2> soubor-chyb
$program > vystup 2> chyb-vystup
$program < vstup > vystup 2> vystup-chyby
```

Chybový výstup a standardní výstup— přesměrování

Chybový výstup lze přesměrovat do standardního výstupu pomocí **2>&1**.

```
$program 2>&1
```

Díky výše uvedenému zápisu je možno standardní výstup a chybový výstup přesměrovat do jednoho souboru.

```
$program > soubor 2>&1
```

Upozornění: nelze obrátit pořadí zápisu, následující zápis je chybný:
`program 2>&1 > soubor`

Pipes

Jednotlivé příkazy lze “zřetězit”. Pomocí *pipes — rour* můžeme výstup jednoho příkazu poslat na vstup druhého příkazu, atd...
Zápis `program1 | program2` znamená: vykonej program `program1` a výstup z tohoto programu předej programu `program2` na vstup. Konstrukci lze tedy přepsat následovně:

```
$program1 | program2
$
$program1 > /tmp/vystup-program1
$program2 < /tmp/vystup-program1
$rm /tmp/vystup-program1
```

Pattern matching a command substitution

Pattern Matching

Při práci v terminálu (pracujeme se shellem) existují speciální znaky, které expandují použijeme-li je v příkazovém řádku. Shell před spuštěním tyto speciální znaky expanduje vzhledem k souborům které se nachází na dané cestě.

Speciální znaky:

- `?` jeden libovolný znak
- `*` libovolný počet znaků (i nulový)
- `[ ]` libovolný znak obsažený v závorce

Příklady: `rm -rf *; touch a?b; cp a[abc]c ~`

Command Substitution

Umožňuje výstup příkazu nahradit původní příkaz. Existují dva druhy zápisů:

- ‘příkaz‘ — starší zápis, funguje ve většině shellů
- $(příkaz) — novější způsob zápisu

Shell spustí příkaz uvedený mezi zpětnými apostrofy, či v závorce za dolarem a výstupem z příkazu nahradí příkaz.

Např:

```
mkdir prednaska-$(date -I)
touch $(seq 20)
```

Řazení obsahu souboru — `sort`

Příkaz **sort** čte data na vstupu a vypisuje data lexikograficky seřazena.

```
$sort soubor.txt
```

Příkaz `sort` má řadu užitečných parametrů:

- -b ignoruje úvodní mezery
- -k `-k=POS1[,POS2]` řadí dle klíče na pozici POS1, atd.
- -n při řazení řadí výstup “číselně”
- -r výpis v opačném pořadí
- -t separátor polí

Např:

```
sort -k 3 -t : -n /etc/passwd | less
```

Řazení obsahu souboru — `sort`, `uniq`

Např:

```
$sort /etc/passwd
$sort -r /etc/passwd
$sort -k 3 -t : -n /etc/passwd | less
```

Ve spojení s příkazem **sort** se často používá příkaz **uniq**. Příkaz **uniq** čte data ze vstupu a pokud řádky bezprostředně za sebou jsou stejné, nechá pouze jednu řádku. Velmi často se program `uniq` využívá s parametrem pro výpočet počtu opakujících se řádek `-c`.

```
$uniq soubor
$sort soubor | uniq
```

Archivace souborů

Vytváření archívů — `tar`

Při archivování souborů (např. vytváření archivu neprázdného adresáře) se používá příkaz **`tar`**. Příkaz **`tar`** vytváří *jeden* archivní soubor, tento soubor není komprimován.

Struktura příkazu má následující tvar

```
tar parametry soubor [soubory]
```

Parametry:

- **c** vytvoří archív se jménem *soubor*
- **v** vypíše na standardní výstup veškeré akce které příkaz provádí
- **x** rozbalí/obnoví soubory z archivu
- **t** vypíše obsažené soubory v archivu
- **f** pokud za parametrem **f** následuje jméno souboru, bude příkaz `tar` operovat nad tímto souborem.

Práce s archívy — `tar`

- **Vytvoření archivu** - archivy vytvořené programem **`tar`**, obvyklá koncovka je **.tar**:

```
tar cvf archiv.tar zdrojove_soubory
```

- **Extrahování archivu** - soubory z archivu obnovíme:

```
tar xvf archiv.tar
```

- **Vylistování obsahu archivu** - výpis souborů které v daném archivu jsou obsaženy:

```
tar tf archiv.tar
```

- **Standardní výstup** - pokud je místo souboru uveden znak **-** pracuje program se standardním výstupem či vstupem (dle toho kde je mínus uvedeno)

```
tar cvf - * | split -b 20k - archiv-$(date -I)
```

Komprimace souborů — `gzip`, `bzip2`

Pro komprimaci souborů v operačním systému GNU / Linux se používají nejčastěji (výhradně ??) programy **`gzip`**, **`bzip2`**. Obvyklé použití při archivaci více souborů je vytvořit tar archiv a tento archiv následně komprimovat. Příkazy **`gzip`**, resp. **`bzip2`** zdrojový soubor zabalí a následně smaží - pokud pomocí parametrů toto defaultní chování nezměníme.

Použití

```
gzip soubor
bzip2 soubor
```

Program **`gzip`**, resp **`bzip2`** vytvoří komprimovaný soubor s příponou **`gz`**, resp. **`bz2`**. Pro rozbalení komprimovaných souborů se používají programy **`gunzip`** a **`bunzip2`**.

Vyhledávání souborů

Vyhledávání souborů — `find`

Příkaz `find` prochází soubory a adresáře v zadaném adresáři a vyhledává soubory dle kritérií.

Struktura příkazu má následující tvar

```
find cesta -name jmeno_souboru
```

Příklad:

```
find /etc -name sources.list
```

Příkaz find umí s využitím parametru `exec` vykonávat nad vyhledanými soubory operace. Za parametr `exec` se doplní příkaz, kde {} reprezentuje aktuálně nalezený soubor a `;` konec operace.
Příklad:

```
find /etc -name motd -exec echo soub '{}' \;
```

Vyhledávání souborů — `find`

Příkaz `find` je schopen vyhledávat soubory i dle dalších kritérií, ukážeme si pouze některé:

dle typu souboru — pomocí přepínače `-type typ`, pro běžné účely se nejčastěji používají `typy`:

- `d` — adresář
- `f` — "klasický" soubor
- `l` — symbolický link

dle vlastníka `-user user_name`

dle práv souboru `-perm maska` Práva jsou zapisována ve stejném formátu jako pro příkaz `chmod`.

dle velikosti `-size velikost` pokud je před velikostí umístěn znak `+`, resp. `-` vyhledáváme soubor větší, resp. menší než je zadaná velikost

dle času viz man stránky

Příklad:

```
find /etc/ -name "a*" -type f -size +2k -user root
```

/dev/random

Práce s WWW v textovém režimu

Existuje řada textových prohlížečů stránek HTML, mezi nejznámější patří programy **lynx** a **linx**.
Použití:

```
lynx  [parametry] URL
links [parametry] URL
```

Tyto textové prohlížeče mají specifické ovládání, více viz. man stránky.

Pro stahování stránek/souborů z www je vhodný řádková program **wget**, tento program stahuje obsah stránky jejíž URL specifikujeme při spuštění. Příkaz **wget** umí stahovat obsah i rekurzivně (parametr -r).

```
$wget [parametry] URL
```

Textový editor EMACS

Textový editor EMACS — viz. cvičení. Zde uvedeme pouze nejpoužívanější klávesové zkratky:

C-x C-c	ukončení práce s editorem
C-x C-f	otevření souboru
C-x C-s	uložení souboru
C-x C-w	uložení bufferu do jiného souboru
C-x i	vložení obsahu jiného souboru do aktuálního bufferu
C-h t	tutoriál
C-s	hledání zadaného řetězce směrem “dopředu”
C-SCP	vybírej od této pozice
C-w	vyjmi region
M-w	kopíruj region
C-y	vlož na aktuální pozici poslední vyjmutý region

Stream editor `sed`

`sed` — stream editor

Aplikace `sed` je *stream editor* pro modifikaci vstupního proudu. Příkaz `sed` má následující syntaxi:

```
sed [volby] 'skript' [vstupni soubor]
```

Stream editor `sed` rozumí řadě příkazů, nejčastěji používaným příkazem je nahrazování znaků ve formátu:

```
sed 's/vzor/novy/flag' soubor
```

Kde jednotlivý význam parametrů je

- `s` název příkazu — nahrazovací příkaz
- `/` oddělovač
- `vzor` vzor je obvykle regulární výraz který se vyhledává
- `novy` nahrazení nalezeného vzoru
- `flag` specifikace co provádět v případě, že na řádku je více než jeden výskyt vzoru

sed — stream editor

Pokud flag není uveden, je nahrazen pouze první nalezený vzor na řádce. V případě že je uveden znak g jsou nahrazeny všechny výskyty vzoru. Při použití čísla je nahrazen právě n-tý výskyt, čísla lze kombinovat se znakem g.
Příklad:

```
echo "ahoj ahoj ahoj ahoj" | sed 's/ahoj/cau/'
echo "ahoj ahoj ahoj ahoj" | sed 's/ahoj/cau/g'
echo "ahoj ahoj ahoj ahoj" | sed 's/ahoj/cau/2'
echo "ahoj ahoj ahoj ahoj" | sed 's/ahoj/cau/2g'
```

V případě že do nového řetězce hodláme vložit řetězec který odpovídá vzoru, použijeme v nahrazení znak &.
Příklad:

```
echo "toto je pokus" | sed 's/.*/::& &::/'
```

`sed` — flag print

Defaultně `sed` tiskne každou řádku (pokud byla provedena náhrada textu, je tištěna modifikovaná řádka). Pokud se příkaz `sed` spustí s parametrem `-n`, flagem `p` explicitně označíme tisk řádky.

```
cat /etc/passwd | sed -n 's/.*/&/p'
```

Stream editor `sed` umožňuje specifikovat rozsah, na kterém se daná operace bude provádět. Pokud před uvedení příkazu umístíme rozsah, bude `sed` provádět daný příkaz pouze v uvedeném rozsahu.
Rozsahy můžeme definovat:

- danou řádku číslem: `'3 s/.*/x&x/'`
- rozsah řádek (od-do): `'1,100 s/.*/x&x/'`
- řádky obsahující daný vzor `'/student/ s/.*/x&x/'`
- rozsahem řádek mezi dvěma vzory `'/student/,/ucitel/ s/.*/x&x/'`

Pokud použijeme v rozsahu znak $, representuje konec souboru.

sed — příkaz delete

Dalším příkazem po příkazu `s` (search), který si uvedeme je příkaz mazání řádku. Tento příkaz je representován znakem `d` a je velmi často používán ve spojení s rozsahy:

```
sed '10   d' <file
sed '10,$ d' <file
sed '/^[0-9]/ d'
```

Rozsahy lze použít samozřejmě i s funkcí `print`:

```
sed '/^$/ p'
sed -n '20,30 p' <file
sed -n '/Dobry den/,/S pozdravem/ p'
```

AWK — pattern scanning and text processing language

AWK — základy

Awk je jazyk určený na práci s textem. Awk lze spustit s řadou parametrů:

```
awk [-F value] [-v var=value] 'program-text' [file ...]
awk [-F value] [-v var=value] [-f prgm-file] [file ...]
```

kde jednotlivé parametry znamenají:

- -F nastavení oddělovače polí
- -v nastaví jednotlivé proměnné
- -f soubor který obsahuje “awk program”

Program `awk` prochází souborem řádek po řádce a pro řádky které odpovídají *vzoru* vykoná příslušnou akci.

AWK — syntaxe

Syntaxe programu `awk` vypadá následovně:

```
awk 'BEGIN    {inicializace}
     vzor     {...}
     vzor     {...}
     vzor     {...}
     END      {závěrečná akce}' soubor
```

Vše co je uvedeno za klíčovým slovem `BEGIN`, resp. `END` je vykonáno na začátku resp. na konci programu. Jednotlivé řádky souboru jsou analyzovány a jestliže řádek odpovídá vzoru, je vykonána nad řádkem příslušná akce. Pokud není nutné vykonat na počátku či konci speciální akce, je možno `BEGIN`, resp. `END` vynechat.

AWK — vzory

Ve vzorech pro program `awk` lze použít pouze jednoduché řetězce či regulární výrazy, tedy vzory jsou podobné jako vzory pro program `grep`, pouze jsou uzavřeny mezi znaky `/`:

- **`/linux/`** - provede akci na řádce obsahující slovo linux.
- **`/^linux/`** - provede akci na řádce začínající slovem linux.
- **`/linux$/`** - provede akci na řádce končící slovem linux
- **`/[Ll]inux/`** - provede akci na řádce obsahující slova Linux či linux.
- **`/(Linux|Java)/`** - provede akci na řádce obsahující slova Linux či Java
- **`/^[+-]?[0-9]+$/`** - provede akci na řádcích obsahující pouze celá čísla

AWK — akce

Možnosti jazyka programu `awk` jsou poměrně rozsáhlé, pro základní práci stačí znát následující:

- Každý řádek lze rozložit na pole dle separátoru, jednotlivé pole:`$1, $2, $3, ...`
- Existují vestavěné proměnné `NR` — číslo aktuální řádky, `NF` — počet polí na aktuální řádce.
- Existují klíčová slova pro cykly a podmínky: `for`, `while`, `if`.
- Pro tisk na standardní výstup se používají příkazy `print` a `printf`.
- Nad proměnnými obsahující čísla lze provádět základní matematické operace.

AWK — ukázka

```
awk -F: '{print $1}' /etc/passwd
awk -F: '{print $1 $2 NR}' /etc/passwd
awk -F: '{if ($3 > 1000) print $1}' /etc/passwd
awk '{if (NR > 100 && NF > 4) print $5}' data.txt
```

AWK — ukázka

```
BEGIN { st1 = 0; pt1 = 0;
        st2 = 0; pt2 = 0;
        st3 = 0; pt3 = 0; }
{
   ...
}
END {
      print "T1\nPsalo: "pt1",celkem: "st1", prumer: "st1/pt1;
      print "T2\nPsalo: "pt2",celkem: "st2", prumer: "st2/pt2;
      print "T3\nPsalo: "pt3",celkem: "st3", prumer: "st3/pt3;
    }
```

```
1 Karel Capek          8   X   9
1 Vitezslav Halek      8  10   6
1 Bozena Nemcova       6   1   X
1 Jaroslav Hasek       3   4   6
1 Eduard Bass          9   9   9
2 Josef Capek          1   2   1
2 Vladislav Vancura    4   X   X
2 Jaroslav Vrchlicky   5   6   7
2 Franz Kafka          2   2   2
```

AWK — ukázka pokračování

```
if ($4 != "X")
   {
    st1 += $4;
    pt1 ++;
   }

if ($5 != "X")
   {
    st2 += $5;
    pt2 ++;
   }

if ($6 != "X")
   {
    st3 += $6;
    pt3 ++;
   }
```

Úvod do programování shellu

Úvod do programování shellu — `bash`

Uživatel komunikující se systémem prostřednictvím “příkazové řádky” využívá příkazového interpretru, tzv. shellu.

Hlavní funkcí příkazového interpretru je spouštět jednotlivé příkazy zadané prostřednictvím standardního vstupu, či pomocí souboru.

Existuje celá řada shellů: `bash`, `csh`, `ksh`, `zsk`. My se budeme zabývat pouze (zřejmě nejčastěji používaným) příkazovým interpretrem `bash`. Základní fakta platí pro všechny interpretry.

Shell — spouštění příkazů

Příkazy se rozdělují:

- **vestavěné** příkazy provádí přímo shell, není potřeba vytváření dalšího procesu — výhoda v rychlosti
- **externí** spustitelné soubory, při spouštění je vytvořen další proces shellu (je vytvořen tzv. *fork*).

Externí příkazy musí mít nastavena příslušná práva (právo spuštění), nezáleží na příponě souboru:

```
chmod +x spustitelny_soubor
```

Jestliže spuštěným souborem binární soubor, shell pustí tento soubor jako potomka. V opačném případě shell předpokládá, že soubor obsahuje jednotlivé příkazy shellu, které mají být provedeny.

Shell — spouštění příkazů II

Při spouštění libovolného příkazu se shell podívá do proměnné `PATH`. V případě že hodláme spustit příkaz který se nenachází ve standardních cestách, je nutno specifikovat plnou cestu k souboru:

```
honza@prasatko:/opt/eclipse$./eclipse
honza@prasatko:/opt/eclipse$/opt/eclipse/eclipse
```

Jestliže shell spouští externí spustitelný soubor, vytvoří svou kopii. Pokud však první řádek spustitelného souboru začíná znaky `#!`, shell následující řetězec považuje za jméno programu který daný skript spustí.

```
#!/bin/bash

echo hello
```

Komentáře, uvozování speciálních znaku

Komentáře v shellu následují vždy po znaku `#`, po kterém je ignorován zbytek řádku.

```
PROM=ahoj    # Nastaveni promenne PROM na ahoj
rmdir pokus # Smazani adresare pokus
```

Existují znaky, které mají speciální význam (stejně tak jak existovaly znaky se speciálním významem v regulárních výrazech). Jedná se o tyto znaky:

```
; & ( ) | < > mezera tabulátor nová řádka
```

Jestliže před tyto znaky místníme znak \ nebudou jako speciální znaky vnímány. Stejného efektu docílíme s použitím páru uvozovek či apostrofů.

Příkaz `echo`

Příkaz `echo` vypisuje na standardní výstup řádek textu. Použití:

```
echo [parametry] TEXT
```

Parametry:

- `-n` při výpisu textu `TEXT` nepřidává na konec automaticky přechod na novou řádku
- `-e` volba umožní používat tzv. “backslash escapes”

Příklady:

```
echo Hello world
echo *
echo $(find . -name ahoj)
```

Proměnné

Proměnné shellu obsahují znakové řetězce. K proměnné daného jména lze přiřadit hodnotu následujícím způsobem:

```
promenna=nova_hodnota_promenne
```

Upozornění: názvy proměnných jsou senzitivní na velká a malá písmena. V případě že se chceme odvolat na proměnnou v příkazové řádce, či v skriptu použijeme před jméno proměnné znak `$`. Proměnné lze načítat z příkazové řádky také pomocí příkazu `read`.
Příklady:

```
PROM=ahoj
echo $PROM
read JMENO
echo Vase jmeno: $JMENO
```

Proměnné

Shell má k dispozici řadu předdefinovaných proměnných které využívá při běhu, ukažme si některé:

PATH	seznam adresářů oddělených znakem `:` v kterých shell hledá spustitelné soubory
PS1	formát promptu
PS2	formát sekundárního promptu
HOME	domovský adresář

Cykly - cyklus `for`

Cyklus `for` iteruje přes seznam

```
for JMENO SLOVA ...; do PRIKAZY; done
```

Pro každé slovo v seznamu, je nastavena proměnná `JMENO` na hodnotu slova a jsou vykonány příkazy `PRIKAZY`.

```
for POZDRAV in ahoj nazdar cau;
do
   echo $POZDRAV
done

for I in $(seq 100);
do
   echo element - $I
done
```

Pokud je v cyklu `for` použito klíčové slovo `break`, je cyklus ukončen.

Cykly - cyklus `while`

Dokud výsledek poslední operace `PRIKAZY` je nulový, prování `AKCE`.

```
while PRIKAZY; do AKCE; done
```

```
while touch /tmp/A;
do
 sleep 10;
done
```

```
COUNTER=5
while [ $COUNTER -ge 0 ]
do
  echo $COUNTER
  COUNTER=$(( $COUNTER - 1 ))
done
```

Testování podmínek — příkaz `test`

Program `test` zjišťuje typy souborů a porovnává hodnoty. Lze zapsat následujícími způsoby:

```
test VYRAZ
[ VYRAZ ]
```

Následující konstrukce jsou pravdivé v případě, že:

- **`( VYRAZ )`** — VYRAZ je true
- **`( ! VYRAZ )`** — VYRAZ je false
- **`( VYRAZ1 -a VYRAZ2 )`** — VYRAZ1 a VYRAZ2 jsou true
- **`( VYRAZ1 -o VYRAZ2 )`** — alespoň jeden z výrazů je true
- **`RETEZEC1 = RETEZEC2`** — řetězce jsou stejné
- **`RETEZEC1 != RETEZEC2`** — řetězce se nerovnají
- **`CISLO1 -eq CISLO2`** — čísla se rovnají
- **`CISLO1 -ge CISLO2`** — `CISLO1` je větší/rovno číslu `CISLO2`
- **`CISLO1 -gt CISLO2`** — číslo `CISLO1` je větší než `CISLO2`

Testování podmínek — příkaz `test`

Program `test` lze použít i pro testování souborů, následující konstrukce jsou pravdivé v případě, že:

- **`( -f SOUBOR )`** — soubor existuje
- **`( -d SOUBOR )`** — soubor existuje a je to adresář
- **`( -e SOUBOR )`** — soubor existuje a je to "klasický" soubor
- **`( -h SOUBOR )`** — soubor existuje a je to symbolický link
- **`( SOUBOR1 -nt SOUBOR2 )`** — soubor `SOUBOR1` byl modifikován dříve než soubor `SOUBOR2`
- **`( SOUBOR1 -o SOUBOR2 )`** — soubor `SOUBOR1` byl modifikován později než soubor `SOUBOR2`

Podmínka `if`

Dokud výsledek poslední operace `PRIKAZY` je nulový, je proveden kód ve větvi `AKCE`. V opačném případě je proveden kód ve větvi `AKCE2`.

```
if PRIKAZY; then AKCE; else AKCE2; fi
if PRIKAZY; then AKCE; [ elif PRIKAZY2; then AKCE3; ]
... [ else AKCE4; ] fi
```

```
if mkdir /tmp/A;
then
    echo "Adresar neexistoval";
else
    echo "Adresar existoval";
fi
```

V případě, že hodláme testovat výsledek vlastních skriptů, defaultně skript vrací hodnotu poslední operace. V případě použití vestavěného příkazu `exit` můžeme vracet libovolné hodnoty.

Expanze proměnných

Při expanzi proměnných lze využít celou řadu konstrukcí, které mohou modifikovat způsob expanze proměnných. Následující přehled obsahuje nejčastěji používané konstrukce:

- `${PROMENNA}` — hodnota proměnné,
- `${PROMENNA:-HODNOTA}` — jestliže proměnná není nastavena, je použita `HODNOTA`
- `${PROMENNA:offset:delka}` — je vrácena část hodnoty proměnné
- `${PROMENNA#slovo}` — z hodnoty proměnné odstraní začátek odpovídající slovu
- `${PROMENNA%slovo}` — z hodnoty proměnné odstraní konec odpovídající slovu,