# Tahák na UOS 2008/2009

*sestavil Tomáš „Inza“ Jukin – http://www.dvojmo.cz*

## OBSAH

OBSAH ..... 1
TYPOGRAFICKÁ POZNÁMKA ..... 2
NAZEV_PRIKAZU ..... 2
SPECIÁLNÍ ZNAKY ..... 2
SPOJOVÁNÍ PŘÍKAZŮ ..... 2
PŘÍKAZY – PRÁCE S PROSTŘEDÍM ..... 3
TYPE ..... 3
WHICH ..... 3
WHO ..... 4
WHOAMI ..... 4
WHEREIS ..... 4
DATE ..... 4
PŘÍKAZY – NAVIGACE ..... 4
CD ..... 4
LS ..... 4
PWD ..... 5
MKDIR ..... 5
RMDIR ..... 5
CP ..... 5
MV ..... 5
RM ..... 6
LN ..... 6
PŘÍKAZY – ZÁKLADNÍ FILTRY ..... 6
CAT ..... 6
SPLIT ..... 6
HEAD ..... 6
TAIL ..... 6
HEAD + TAIL ..... 7
CUT ..... 7
PASTE ..... 7
CUT + PASTE ..... 7
WC ..... 7
LESS ..... 7
MORE ..... 8
PŘÍKAZY – POKROČILÉ FILTRY ..... 8
SORT ..... 8
UNIQ ..... 8
TEE ..... 8
TR ..... 8
GREP ..... 8
FGREP ..... 9
EGREP ..... 9
CMP ..... 9
COMM ..... 9
DIFF ..... 10
PATCH ..... 10
FIND ..... 10
PŘÍKAZY – PROGRAMOVATELNÉ FILTRY ..... 11
SED ..... 11
AWK ..... 11
PŘÍKAZY – ADMINISTRACE ..... 13
CHOWN ..... 13
CHMOD ..... 13
TAR ..... 13
UNZIP ..... 13
EXEC ..... 13
NICE ..... 14
RENICE ..... 14
KILL ..... 14
NOHUP ..... 14
PS ..... 14
PRSTAT ..... 15
PTREE ..... 15
LAST ..... 15
PŘÍKAZY – PROGRAMOVÉ STRUKTURY ..... 15
PROMĚNNÉ: ..... 15
POLE: ..... 15
PŘÍKAZ IF: ..... 15
PŘÍKAZ CASE: ..... 16
PŘÍKAZ WHILE: ..... 16
PŘÍKAZ FOR: ..... 16
TEST, [ ... ] ..... 16
EXPR ..... 17
LET NEBO (( )) ..... 17
SHIFT ..... 17
READ ..... 17
POMŮCKY PRO LADĚNÍ: ..... 18
OSTATNÍ ..... 18

***Poznámka:***

*Informace uvedené v tomto textu NEJSOU úplným popisem, či přepisem manuálových stránek. Účelem tohoto textu není zahltit čtenáře informacemi. Tento text obsahuje základní popisy všech důležitých příkazů, probíraných v předmětu Y36ALG na ČVUT – FEL. Tento text by měl sloužit jako studijní materiál k testům z předmětu. Informace, které nejsou zde, by se neměly objevit ani v testech.*

# Tahák na UOS 2008/2009

*sestavil Tomáš „Inza“ Jukin – http://www.dvojmo.cz*

## TYPOGRAFICKÁ POZNÁMKA

### nazev_prikazu

*(jak vznikl název) krátký popis příkazu*

```
obecny zapis prikazu [nepovinne casti]
```

**Dlouhý popis příkazu. Zde je napsáno, co příkaz dělá a jak funguje.**

*Volby:*

- *-s tichy rezim prikazu*
- *-v ukecany rezim prikazu*
- *-f soubor ze kterého se ma cist*

*code:*

```
# ukazka_prikazu -f parametr1 'parametr2' "parametr3"
```

```
Př.: Zadání vyřešeného příkladu použití příkazu (příklad 1):
# ukazka_prikazu -f parametr1 'parametr2' "parametr3"
```

```
Př.: Zadání vyřešeného příkladu použití příkazu (příklad 2):
# ukazka_prikazu -f parametr1 'parametr2' "parametr3" \
> krery je na vice radek
```

***Poznámka:***

*Poznámka k příkazu – pokud je nějaká*

## SPECIÁLNÍ ZNAKY

**BASH na své příkazové řádce zpracovává následující speciální znaky:**

```
` `     = zavolá to, co je uvnitř jako nový příkaz a vrátí výsledek,
          použitý symbol je obrácený apostrof(!) - na klávesnici je pod
          ESC
' '     = zobrazí 100% vstup (nebere v potaz $ ani "") je nutné užít \'
          pro zobrazeni '
" "     = preloží $prom, `call`
$( )    = ` `
(( ))   = vyhodnotí aritmetický výraz a vrátí výsledek
$(( )) = vypíše výsledek aritmetického výrazu
```

**Dále provádí nahrazení těchto znaků:**

```
~       = je nahrazeno za domovský (home) adresář aktuálního uživatele
~user   = je nahrazeno za domovský (home) adresář uživatele „user“
*       = je nahrazeno za všechny položky v daném umístění
```

## SPOJOVÁNÍ PŘÍKAZŮ

**V BASHi lze příkazy spojovat 3 způsoby:**

- *Sekvenčně (pomocí „;“):*

*code:*

```
        # prikaz1; prikaz2
```

*Příkazy jsou provedeny jako by byly zadány postupně pomocí klávesnice, jeden po druhém, zcela nezávisle na sobě*

```
        Př.: Vlezte do domovského adresáře a zobrazte jeho obsah:
        # cd; ls
```

- *Selekčně (podmíněně):*

*code:*

```
        # prikaz1 && prikaz2
        # prikaz1 || prikaz2
        # prikaz1 && prikaz2 && prikaz3
```

*Příkazy jsou provedeny postupně, ale jsou na sobě závislé. Provedení každého příkazu zde závisí na provedení předcházejícího. Pokud jsou dva příkazy spojeny znakem „&&“, provede se následující jen tehdy, pokud byl předcházející úspěšný. Jsou-li příkazy spojeny znakem „||“, provede se následující jen tehdy, pokud předcházející selhal.*

```
        Př.: Vypište OK, pokud se zdařilo ls v aktuální adresáři:
        # ls && echo "OK"
```

- *Paralelně (rourou, pipou, v koloně):*

*code:*

```
        # prikaz1 | prikaz2
        # prikaz1 | prikaz2 | prikaz3
```

*Příkazy běží najedou paralelně vedle sebe. Výstup (standardní výstup - 1) prvního je vstupem dalšího. Nejpoužívanější způsob práce s BASHem. Velmi silná zbraň. Jen, aby bylo jasno v terminologii: roura, či pipa (angl. pipe) je označení pro dva příkazy, spojené znakem „|“, pokud je příkazů takto za sebou, říkáme tomu „kolona“.*

```
        Př.: Vypište počet souborů a adresářů v aktuálním adresáři:
        # ls | wc -l
```

**Všechny výše uvedené způsoby lze vzájemně kombinovat:**

Př.: Vypište počet položek v aktuálním adresáři, když skončí wc chybou, napište „Chyba wc“, poté vypište výzvu „Zadejte další příkaz“:

```
# ls | wc -l || echo "Chyba wc"; echo "Zadejte další příkaz"
```

**Poslední mocnou zbraní, kterou nám BASH dovoluje používat je přesměrovávání:**

*Každý příkaz (příkaz BASHe, externí program, či skript) má na UNIXu automaticky 3 věci:*

- *Standardní vstup – deskriptor 0*
- *Standardní výstup – deskriptor 1*
- *Standardní chybový výstup – deskriptor 2*

*Když nějaký příkaz spustíme (samostatně), tak mu BASH automaticky namapuje (přiřadí) deskriptory následovně:*

- *Na standardní vstup(0) je napojen vstup z klávesnice*
- *Na standardní výstup(1) je napojena obrazovka*
- *Na standardní chybový výstup(2) je napojena obrazovka*
- *Pokud příkazy spustíme v koloně, propojí je BASH mezi sebou.*
- *My ale můžeme také explicitně (ručně) výstup ze skriptů přesměrovat:*
- *>* – *přesměruje standardní výstup(1), pokud cíl už existuje, přepíše ho*
- *>>* – *opět přesměruje standardní výstup(1), ale nepřepisuje, jen přidává*
- *<* – *nasměruje do příkazu nějaký soubor jako vstup.*
- *<<END* – *nasměruje do souboru vstup z klávesnice, který končí výskytem END na samostatném řádku (místo END můžeme použít co chceme)*

Př.: Vypište počet položek v aktuálním adresáři a uložte do souboru adr.txt:

```
# ls | wc -l > adr.txt
```

Př.: Vypište počet položek v aktuálním adresáři a přidejte do souboru adr.txt:

```
# ls | wc -l >> adr.txt
```

Př.: Načtěte obsah z klávesnice a uložte do souboru file.txt:

```
# cat > adr.txt <<END
```

Kromě výše uvedeného umí BASH přesměrovávat i podle čísla deskriptoru a reltivně (takto můžeme přesměrovat naPř.: chybový výstup, nebo prohodit výstupy mezi sebou):

Př.: Načti obsah aktuálního adresáře, ulož standardní výstup do stdout.txt a chybový do stderr.txtt:

```
# ls >stdout.txt 2>stderr.txt
```

Př.: Zahoď chybový výstup, standardní normálně vypisuj:

```
# ls 2>/dev/null
```

Př.: Zahoď standardní výstup a pošli chybový tam, kam standardní:

```
# ls >/dev/null 2>&1
```

Př.: Prohoď standardní vstup a výstup:

```
# ls 3>&1 1>&2 2>&3
```

## PŘÍKAZY – PRÁCE S PROSTŘEDÍM

*type, which, who, whoami, whereis, date*

### type

*(command TYPE) vyhledá spustitelný soubor skriptu v shellové cestě*

```
type param1 [param2 […]]
```

**Podobně jako příkaz which, vyhledá spustitelný soubor zadaného skriptu/programu/příkazu shellu**

*code:*

```
# type man
# type which
```

***Poznámka:***

*Tento příkaz je přímo zabudován v BASHi, je rychlejší než which*

### which

*(WHICH command is this) vyhledá spustitelný soubor skriptu v shellové cestě*

```
which param1 [param2 […]]
```

**Vyhledá spustitelný soubor zadaného příkazu v cestě (PATH), je to samostatný program, je pomalejší než type**

*code:*

```
# which who
```

***Poznámka:***

*Je pomalejší než type, používat type je lepší. Pokud se c cestě vyskytne několik souborů stejného jména, vypíše se první z nich*

# Tahák na UOS 2008/2009

*sestavil Tomáš „Inza“ Jukin – http://www.dvojmo.cz*

### who

*(WHO is logged in) vypisuje právě přihlášené uživatele s informacemi o jejich relacích*

```
who
```

**Vypíše seznam přihlášených uživatelů s informacemi o loginu, terminálu, času poslední aktivity a připojení**

*code:*
```
# who
```

***Poznámka:***

*Pokud vás zajímá jen váš login, použijte whoami*

### whoami

*(WHO AM I) vypíše přihlašovací jméno aktuálního uživatele*

```
whoami
```

**Vypíše jméno aktuálně přihlášeného uživatele**

*code:*
```
# whoami
```

### whereis

*(WHERE IS this) zkouší nalézt dané soubory pomocí databáze*

```
whereis param1
```

**Zkouší nalézt soubor(y) pomocí indexu. Není spolehlivé, neboť není zaručeno že bude index aktuální.**

*code:*
```
# whereis who
```

***Poznámka:***

*Příkaz nemusí být nainstalován*

### date

*(show DATE) vypíše aktuální datum a čas*

```
date ["+%H:%M:%S"]
```

**Zobrazí aktuální datum a čas**

*Volby:*

- *"+%H%M%S" – formát výstupu, více viz. „man -s 3C strftime“*

*code:*
```
# date
# date "+%H:%M:%S"
```

***Poznámka:***

*Další formátování: %Y, %M, %D, %W, …*

## PŘÍKAZY – NAVIGACE

*cd, ls, pwd, mkdir, rmdir, cp, mv, rm, ln*

### cd

*(Change Directory) změní aktuální adresář*

```
cd [param1]
```

**Změní aktuální adresář na cestu uvedenou v parametru. Není-li uvedena, změní adresář na domovský adresář aktuálního uživatele**

*code:*
```
# cd
# cd /home/user/ukoly
# cd /etc/passé
```

### ls

*(LiSt directory) zobrazí obsah adresáře*

```
ls [-alFLhi] [param1 [param2 […]]]
```

**Zobrazí obsah adresáře/ů zadaných v parametrech. Nejsou-li uvedeny, vypíše obsah aktuálního adresáře. Defaultně pouze jména**

*Volby:*

- *-a zobrazí i „skryté“ soubory – ty začínající tečkou (např. „.bashrc“)*
- *-l dlouhý výpis, zobrazí nejen jména, ale také atributy (velikost, práva, vlastníka, skupiny, typ, …)*
- *-F výpis včetně indikace typu, za adresáře přidává „/“, atd…*
- *-L zobrazí pouze cíle symbolických linků*
- *-h human-friendly forma – zobrazuje velikosti v jednotkách KB, MB a GB*
- *-i zobrazí čísla i-nodů*

*code:*
```
# ls
```

```
# ls -al
```

## pwd

*(Personal Working Directory) zobrazí cestu k aktuálnímu adresáři*

```
pwd
```

**Zobrazí cestu k aktuálnímu adresáři**

*code:*

```
# pwd
```

## mkdir

*(MaKe DIRectory)vytvoří adresář*

```
mkdir [-p] [-m 0722] param1
```

**Vytvoří adresář na místě zadaném cestou v parametru. Pokud není použita volba –p, pak musí nadřazené adresáře již existovat**

*Volby:*

- *-p vytvoří nejen poslední, ale všechny adresáře v cestě*
- *-m dovoluje specifikovat oktalové číslo (masku) pro práva k adresáři, implicitně se práva nastavují dle shellové proměnné UMASK*

*code:*

```
# mkdir adr
# mkdir -p /home/web/sites
# mkdir -m 0722
```

## rmdir

*(ReMove DIRectory) smaže prázdný adresář*

```
rmdir [-p] param1
```

**Smaže prázdný(!) adresář uvedený v parametru. Pokud je uvedena volba –p, smaže i rodičovské v cestě (jsou-li prázdné)**

*Volby:*

- *-p maže i s cestou*

*code:*

```
# rm adr
# rm -p /home/web/sites
```

***Poznámka:***

*Pro mazání adresářů včetně obsahu použijte „rm –R“*

## cp

*(CoPy file) kopíruje soubory z umístění A do umístění B*

```
cp [-Rif] co kam
```

**Kopíruje obsah parametru co do umístění dané parametrem kam**

*Volby:*

- *-R rekurentní kopírování (včetně podadresářů)*
- *-i interaktivní režim (zeptá se před přepsáním)*
- *-f „force“ režim, přepisuje „na férovku“*

*code:*

```
# cp /home/my_file /tmp
# cp -R /home/my_dir/ /dev/null
```

***Poznámka:***

*Pokud má příkaz definován alias s“ –i“ a vy chcete jeho efekt vyrušit, předřaďte volání cp v rouře příkaz „yes“*

## mv

*(MoVe file) přejmenovává/přesunuje soubory*

```
mv [-if] co kam
```

**Přesunuje soubor/adresář/link z umístění definovaného parametrem co do umístění v parametru kam. Pokud provedete přesun v rámci stejného adresáře, dojde k přejmenování (ono totiž přejmenování není vlastně nic jiného než přesun ve stejném adresáři)**

*Volby:*

- *-i interaktivní režim (zeptá se před přepsáním)*
- *-f „force“ režim, přepisuje „na férovku“*

*code:*

```
# mv /home/web /home/web/novy_web
# mv stare_jmeno nove_jmeno
```

***Poznámka:***

*Pokud má příkaz definován alias s“ –i“ a vy chcete jeho efekt vyrušit, předřaďte volání mv v rouře příkaz „yes“*

# Tahák na UOS 2008/2009

*sestavil Tomáš „Inza“ Jukin – http://www.dvojmo.cz*

### rm

*(ReMove file) odstraňuje soubory*

`rm [-Rif] co`

**Odstraní (smaže) soubor/adresář zadaný v parametru co**

*Volby:*

- *-R rekurentní režim (mazání včetně podadresářů)*
- *-i interaktivní režim (zeptá se před přepsáním)*
- *-f „force“ režim, přepisuje „na férovku“*

*code:*
```
# rm -R /home/web
# rm nejaky_soubor
```

***Poznámka:***

*Pokud má příkaz definován alias s“ –i“ a vy chcete jeho efekt vyrušit, předřaďte volání rm v rouře příkaz „yes“*

### ln

*(create LiNk) vytvoří odkaz (link)*

`ln [-s] cil odkaz`

**Vytvoří odkaz na nějaký soubor na disku. Odkazy mohou být „měkké“ (obdoba „zástupce“ z MS Windows) a tvrdé (více dveří do jedné místnosti). Měkké odkazy mohou ukazovat na adresáře i jiné disky. Tvrdé nikoliv. Tvrdý odkazy na adresář může vytvořit pouze superuživatel**

*Volby:*

- *-s vytvoří měkký (symbolický) odkaz – symlink*
- *-d vytvoří tvrdý odkaz na adresář (smí jen root)*

*code:*
```
# ukazka_prikazu -f parametr1 'parametr2' "parametr3"
```

***Poznámka:***

*Kam ukazuje cílový soubor můžeme zjistit např. pomocí těchto příkazů („readlink“, „ls –l“)*

## PŘÍKAZY – ZÁKLADNÍ FILTRY

*cat, split, head, tail, cut, paste, wc, less, more*

### cat

*(conCATenate) kopíruje stdin, spoji ho, vypíše na stdout*

`cat [muj_soubor]`

**Spojí vstup a zkopíruje na výstup, využívá se ke čtení souborů a často jako počáteční bod roury**

*code:*
```
# cat /etc/passwd
# cat xa? > file.orig
```

*Volby:*

- *-n přidá čísla řádek*

### split

*(SPLIT file) rozdělí vstup na části*

`split [-l X] [-b 10k] co`

**Rozdělí vstup (parametr co) na části (po bytech nebo řádcích) a uloží do nameaa, nameab...**

*Volby:*

- *-b 10k po 10kB*
- *-l 123 po 123 řádkách*
- *-a 4 dělat 4char přípony*

*code:*
```
# split -l 100 velky_soubor
```

***Poznámka:***

*Opětovné spojení pomocí „ cat xa? > file.orig“*

### head

*(HEAD of file) zobrazuje řádky od začátku souboru*

**Zavolá-li se bez volby, zobrazí prvních 10 řádek z výstupu**

```
Př.: Zobrazí prvních 13 řádek:
# head -13 [file]
# head -n 13 [file]
```

### tail

*(TAIL of file) zobrazuje řádky od konce souboru*

Př.: Zobrazí posledních 11 řádek:
```
# tail -11 [file]
```

Př.: Zobrazí soubor od 4. řádky:
```
# tail +4 [file]
```

## head + tail

Př.: Vypsat 15. řádek:
```
# head -15 | tail -1
```

Př.: Vypsat řádky 15., 16., 17.:
```
# tail +15 | head -3
```

## cut

*(CUT into parts) rozdělí soubor po sloupcích*

*Volby:*

- *-dX oddělovač bude X*
- *-f3,4 zajímá nás sloupec 3 a 4*

*code:*
```
# cut -d: -f3,5 /etc/passwd
```

## paste

*(PASTE to file) slepí soubor z částí (sloupců)*

*Volby:*

- *-dX oddělovač bude X*

*code:*
```
# paste -d: col1 col5
```

## cut + paste

*rozdělí sloupce do souboru a spojí*

*code:*
```
# cut -d: -f1 > a
# cut -d: -f2 > b
# cut -d: -f3 > c
# paste -d ";" a b c
```

## wc

*(Word Counter) počítá řádky/slova*

`wc [-l] [-w] [-c] [vstup]`

**Spočítá řádky/slova/bajty(znaky) ze vstupu a zobrazí je. Pokud je spuštěn bez voleb, zobrazí vše. Často se používá jako poslední prvek kolony**

*Volby:*

- *-l počet řádků*
- *-w počet slov*
- *-c počet bajtů (znaků)*
- *-L v každém souboru najdi nejdelší řádek a vypiš jeho délku*

*code:*
```
# ls -a | wc -l
```

## less

*(show LESS) prohlížeč dlouhých souborů*

`less [-mNsS] [vstup]`

**Interaktivní prohlížečka dlouhých souborů, často se používá k prohlížení konce kolony**

*Volby:*

- *-m upovídaný prompt*
- *-N číslování řádků*
- *-s „squeeze“ režim, smrskne více mezer v jednu*
- *-S nezalamuje řádky*

*Příkazy:*

- *h, H – zobrazí nápovědu*
- *mezera, f ^V, ^F – dopředu o jednu stránku*
- *enter – dopředu o jeden řádek*
- *b, ^B, ESC –b – dozadu o jednu stránku*
- */ – hledání podle regulárních výrazů, za lomítky napsat regulárek a dát enter, less najde první výskyt*
- *? – totéž co „/“, ale směrem nazpět*
- *n – hledej dál, podle posledně nastaveného parametru*
- *N – jako „n“, ale nazpět*
- *V – edituj obsah implicitním editorem*

*code:*
```
# cat /etc/dict/words | less
```

*Tento příkaz je novější a lepší variantou již zastaralého „more“*

### more

*(show MORE) prohlížeč dlouhých souborů*

```
more
```

**Interaktivní prohlížečka dlouhých souborů, tento příkaz je zastaralý a byl nahrazen mnohem výkonnější a lepší variantou less. Zachází se s ním stejně a podobně se ovládá. Je ale méně výkonný a omezenější.**

***Poznámka:***

*Více informací viz. „less“*

## PŘÍKAZY – POKROČILÉ FILTRY

*sort, uniq, tee, tr, grep, fgrep, egrep, cmp, comm, diff, patch, find*

### sort

*(SORT list) seřadí vstup*

```
sort [-n] [-tX] [-kA,B,..]
```

**Svůj vstup seřadí a pošle na výstup. Příkaz se používá v rouře**

*Volby:*

- *-n řadí numericky, ne jako string*
- *-tX oddělovač sloupců bude „X“*
- *-k3,4 řadí podle sloupce 3., je-li shodný, pak 4.*

*code:*

```
# cat /etc/passwd | sort -t: -k3
```

***Poznámka:***

*Pokud chceme použít příkaz uniq, je před ním vždy nutné použít příkaz sort*

### uniq

*(make it UNIQue) odstraňuje duplicity ze setříděného vstupu*

```
uniq [soubor]
```

**Odstraňuje duplicity ze vstupu. Používá se často v rourách. Vstup musí být setříděný**

*code:*

```
# ls | sort | uniq
```

***Poznámka:***

*pro správnou funkci příkazu musí být vstup setříděn příkazem sort*

### tee

*(TEE crossing) téčková odbočka, svůj vstup posílá na výstup, ale zároveň jej kopíruje*

```
tee [soubor]
```

**Používá se pro diagnostiku roury. Svůj vstup posílá na výstup, a zároveň jej kopíruje (buď do souboru, nebo na výstup s deskriptorem 3)**

*code:*

```
# date | tee datum | wc -l
```

### tr

*(TRanslate) překládá znaky*

```
tr [-s] co cim
```

**používá se v rouře. Nahrazuje znaky z parametru co znaky v parametru cim. Oba parametry MUSÍ být stejné dlouhé.**

*Volby:*

- *-s „squezze“ režim. Více výskytů znaků z parametru co nahradí pouze jedním*

```
Př.: Nahradí a za 1, b za 2, c za 3:
# tr abc 123

Př.: Převede na mala:
# tr '[A-Z]' '[a-z]'

Př.: Rozdělí slova na řádky:
# tr -cs "[:alpha:]" "[\n*]" <file1 >file2
```

***Poznámka:***

*V druhém parametru lze použít hvězdičku (*)*

### grep

*(Globbal search for Regular Expression and Print) hledá ve vstupu regulární výraz a tiskne ho*

```
grep [-v] [-l] regexp
```

**Ve svém vstupu vyhledá řádky, v nichž se vyskytuje regulární výraz zadaný parametrem regexp.**

# Tahák na UOS 2008/2009

*sestavil Tomáš „Inza“ Jukin – http://www.dvojmo.cz*

*Volby:*

- *-v výpis jen těch řádků, co se neshodují s regulárním výrazem*
- *-l vypíše jen názvy souborů, ve kterých je shoda*

*Regulární výrazy:*

- *jsou rekurentní (nahrazují se od nejmenšího výskytu – neboli jakmile se to v textu vyskytne, tak se to nahradí)*
- *.  – jakýkoliv znak*
- *[ab0-9]  – výčet znaků, odpovídá právě jednomu znaku z výčtu*
- **  – předcházející znak je uveden nula, nebo libovolně-krát*
- *\{1,3\}  – předcházející znak je uveden 1-krát až 3-krát*
- *$  – konec řádku*
- *^  – začátek řádku*
- *\<  – začátek slova*
- *\>  – konec slova*
- *\(text\)  – uzavře „text“ do bloku, lze se na něj pak odkazovat*
- *\1  – doplní na místo obsah toho, co bylo nahrazeno v 1. závorce zleva*

*code:*

```
# cat file | grep 'ahoj'
# ls | grep -vl 'tento text to nebude obsahovat'
```

```
Př.: Výpis řádků obsahující dvě stejná slova:
man head | grep -i '\(\<[a-z][a-z]*\>\).*\<\1\>'
znak\{m,n\} - m až n výskytů znaku
```

***Poznámka:***

*závorky v grepu potřeba backslashovat (\), parametry v apostrofech (')*

## fgrep

*(Fast GREP) hledá ve vstupu text a tiskne ho*

**Je stejný jako grep, ale neumí regulární výrazy. Je to rychlejší varianta. Nepoužívá se, páč má malé možnosti**

***Poznámka:***

*více viz. grep*

## egrep

*(Extended GREP) hledá na vstupu rozšířený regulární výraz a tiskne ho*

**Je stejný jako grep, ale pracuje s rozšířenými POSSIXovými regulárními výrazy, které známe např. z C a C++. Od grepu se liší použitými regulárními výrazy. Příliš se nepoužívá kvůli nekompatibilitě**

*Vlastnosti:*

- *Nepodporuje znaky: \(, \), \n, \<, \>, \{, \}*
- *Navíc podporuje znaky: +, ?, |, (, )*
- *RE1|RE2 – nebo*
- *znak+ = 1+*
- *znak* = 0+*

## cmp

*(CoMPare files) porovnává soubory binárně*

```
cmp soubor1 soubor2
```

**Provede binární porovnání souborů, v případě úspěchu nevypíše nic, v případě neúspěchu vypíše první rozdíl. Není v praxi moc použitelný, používá se pouze ke zjištění, zda jsou soubory shodné.**

*Volby:*

- *-s tichy režim příkazu*
- *-l dlouhý výstup, vypisují se všechny rozdíly bajt pop bajtu*

*code:*

```
# cmp mujsoubor tvujsoubor
```

## comm

*(COMpare files) porovnávádva setříděné soubory*

```
obecny zapis prikazu [nepovinne casti]
```

**Porovnává dva setříděné soubory. Vypíše co je v prvním a není ve druhém, co je ve druhém a není v prvním a co je v obou. Ve výstupu jsou tedy tři sloupce oddělené tabulátory**

*Volby:*

- *-1 potlač sloupec 1*
- *-2 potlač sloupec 2*
- *-3 potlač sloupec 3*

*code:*

```
# comm -12 file1 file2
```

***Poznámka:***

*Místo tohoto příkazu se v praxi používá jeho modernější varianta diff*

## diff

*(DIFFerence in files) porovnává soubory po řádcích, vytváří záplaty*

`diff souborA souborB`

**Porovná dva soubory po řádcích. Na výstupu dovede popsat změny v souborech tak, že vytvoří jakýsi popis toho, jak se od jednoho ke druhému souboru došlo. Lze jej tedy použít jako jakýsi primitivní verzovací systém.**

*Volby:*

- *-i case insenzitive (nebere ohled na velikost znaků)*
- *-b ignoruje mezery*
- *-B ignoruje prázdné řádky*

*code:*

```
# diff puvodni novy > fix.txt
```

***Poznámka:***

*Výstup z příkazu diff lze použít jako vstup pro program patch, který je schopen změny aplikovat*

## patch

*(PATCH file) aplikuje záplaty, vytvořené programem diff*

`patch < záplata`

**Aplikuje záplaty (patche) vytvořené programem diff. Není třeba uvádět cíle operace, program je automaticky načte ze souboru záplaty. Zadáním volby –R lze záplatu revertovat (vzít zpět)**

*Volby:*

- *-R revertuje (vezme zpět) záplatu*

*code:*

```
# patch < fix.txt
```

***Poznámka:***

*Tento program lze společně s programem diff využít k vytvoření jakéhosi primitivního revizního systému. Místo toho vám ale vřele doporučuji využít v praxi použitelnější moderní nástroje - jako jsou svn, či git*

## find

*(FIND file) hledá soubory*

**Hledá soubory a adresáře podle specifikovaných vlastností**

```
Př.: Nalezeni všech souboru v adresáři /home/courses/Y36UOS:
# dir=/home/courses/Y36UOS
# find $dir

Př.: Nalezeni všech obyčejných souboru:
# find $dir -type f

Př.: Větších než 1000 bloku:
# find $dir -type f -size +1000

Př.: Menších než 100 bajtu (znaku) :
# find $dir -type f -size -100c

Př.: Nalezeni všech obyčejných souboru mladších než týden:
# find $dir -type f -mtime -7

Př.: Starších než 10 dni:
# find $dir -type f -mtime +10

Př.: S i-nodem číslo 314338:
# find $dir -type f -inum 314338

Př.: S alespoň 1 dalším hardlinkem:
# find $dir -type f -links +1

Př.: Nalezeni všech souboru s nastaveným set-gid bitem:
# find $dir -perm -g+s

Př.: Vypsání jména souboru při každé splněné podmínce:
# find /usr/bin /usr/*/bin \
# -name '*awk' -print \
# -type l -print

Př.: Vypsání detailu nalezených souboru:
# find /usr/bin -name '*grep' -ls

Př.: Spuštěni externího příkazu:
# find /etc -type f -exec grep -l 'Solaris 10' {} \;
# find /etc -type f -exec grep -l 'Solaris 10' {} +

Př.: Spuštěni externího příkazu s dotazem:
# find ~ -type f -size 0 -ok rm {} \;

Př.: Nalezeni adresářů od hloubky 3:
# find . -type d | grep '/.*/.*/'
```

```
Př.: Smazáni prázdných souborů a souboru *.delme mladších než je soubor
asdf:
# find . -newer asdf \
# \( -size 0 -o -name '*.delme' \) \
# -ok rm {} \;
```

## PŘÍKAZY – PROGRAMOVATELNÉ FILTRY

*sed, awk*

### sed

*(Stream Editor) řádkový programovatelný editor*

`sed [-n] [-f vstup] prikazy`

**Tento program zpracovává svůj vstup (předaný přes volbu –f, nebo rourou) a zpracovává jej po řádcích. Neboli, pro každý řádek vstupu provede příkazy předané v parametru prikazy. V tomto parametru může být uvedeno libovolné množství příkazů, oddělených středníky. Příkazy se uvádějí ve tvaru:**

`'podminkaAKCE'`

**kde podminka je podmínka, která musí být splněna před provedením AKCE. sed je řádkový editor a proto má podmínka tento formát:**

`radekOD,radekDO`

**kde uvedené proměnné jsou čísla řádek, nebo regulární výrazy ohraničené v „/“ a „/“, a nebo speciální znak $ pro poslední řádek. Řádky jsou číslovány od 0. Není-li uveden radekDO, projede se až do konce souboru. Není-li uvedeno ani radekOD, projede se celý soubor. Příkaz je jeden ze 4 příkazů uvedených níže. sed defaultně to, co nesmažeme vypisuje. Toto chování se mění pomoví volby „–n“. Výsledný formát celé příkazové části tedy vypadá takto:**

`'odkud1,kam1[dpq(s…)][;odkud2,kam2[dpq(s…)]]'`

*Volby:*

- *-n tiskne pouze přikázané přes příkaz p*
- *-f vstupní soubor*

*Příkazy:*

- *d – zruší řádku*
- *p – tiskne řádku*
- *q – skončí*
- *s/re1/re2/volby – nahradí re1 za re2, lze využívat regulární výrazy a reference (\1, \2, …)*

```
Př.: Vytisknout řádky 2-4:
$ sed -n '2,4p' data.txt

Př.: Vytisknout od 4 do konce:
$ sed -n '4,$p' data.txt

Př.: Vytiskne řádky nezačínající na J:
$ sed -n '/^J/p' data.txt

Př.: Vytisknout řádky od r. končicího na 38 po řádek končicí na 27:
$ sed -n '/38$/,/27$/p' data.txt

Př.: Náhrada:
$ sed 's/Praha/Louny/' data.txt

Př.: Nahradit dvoučíslí na konci řádky za nej + " let":
$ sed 's/[0-9][0-9]$/& let/' data.txt

Př.: Výpis souboru až po první prázdný řádek:
sed '/^$/q' /etc/init.d/nfs.server

Př.: Náhrada prvního slova man na řádku za !!man!!:
man man | sed 's/\<man\>/!!man!!/'

Př.: Náhrada druhého slova man na řádku za !!man!!:
man man | sed 's/\<man\>/!!man!!/2'

Př.: Náhrada všech slov man na řádku za !!man!!:
man man | sed 's/\<man\>/!!man!!/g'

Př.: Výpis pouze řádků, kde došlo k náhradě man za !!man!!:
man man | sed -n 's/\<man\>/!!man!!/gp'

Př.: Výpis kde došlo k náhradě bez ohledu na velikost písmen:
man man | sed -n 's/\<[Mm][Aa][Nn]\>/!!man!!/gp'
```

### awk

*(alfred v. Aho, peter j. Weinberger a brian w. Kernighan) programovací jazyk textových manipulací*

`awk program [soubor]`

**awk je programovací jazyk pro filtrování textu. Jak o každý programovací jazyk, tak i awk musí k práci mít svůj program. Ten mu předáme v jeho prvním parametru. Může to být přímý vstup (zde často využívám operátor <<), nebo soubor. Jako druhý parametr můžeme uvést vstupní data, nebo lze awk umístit do roury.**

**Program pro awk sestává z posloupnosti řádků:**

**'vzor' {akce}**

**kde vzor je regulární výraz (potom musí být v „/“ a „/“), či podmínka a akce zastupuje příkaz(y). Výchozí akce je výpis. Výchozí vzor (podmínka) je pravda (provede pro celý soubor). Existují speciální vzory BEGIN a END:**

- *BEGIN { akce } – Provede příkazy akce na začátku běhu skriptu ještě před tím, než jsou zpracována vstupní data*
- *END { akce } – Podobně jako v předchozím případě, ale akce se provede až na konci běhu skriptu*
- */vzor/ – Vypíše všechny řádky vyhovující vzoru (regulární výraz)*
- *{ akce } – Provede akci pro každý vstupní řádek*

**Hlavní zbraní awk je možnost práce s jednotlivými sloupci a řádky vstupu. awk pracuje tak, že soubor nejprve rozdělí na tzv. recordy (záznamy), pomocí separátoru RS a poté s nimi nakládá jako s jednotlivými řádky (obdobně jako sed), pro každý z nich provede ověření, zda odpovídá některému ze vzorů a poté provede akci k němu vázanou.**

**Každý record (řádek) rozdělí na položky (fields) podle separátoru FS. Na tyto položky se poté můžeme odkazovat pomocí $X, kde X je celé číslo sloupečku od 1 zleva. V $0 je uložen celý rekord (řádek).**

*Volby:*

- *-F: specifikuje „:“ jako FS (oddělovač sloupců – field separator)*

*Proměnné:*

- *RS – Record Separator, odděluje řádky, defaultně „\n“, neměňte*
- *FS – Field Separator, odděluje sloupce, defaultně „ “, s ním se často pracuje*
- *ORS – Output Record Separator, odděluje recordy na výstupu, defaultně „\n“, neměňte*
- *OFS – Output Field Separator, odděluje sloupce na výstupu, defaultně „ “, často se specifikuje*
- *NR – Number of Record, číslo aktuálního recordu (řádku v souboru)*
- *NF – Number of Field, číslo aktuálního sloupečku*
- *FILENAME – obsahuje název aktuálního vstupního souboru, „-“ v případě stdin (z roury)*

*Příkazy:*

- *print něco něco něco – základní příkaz k výpisu*
- *printf("%d", FS) – formátovaný výpis, jako v C, nebo Systém.Out.printf() v Javě*

*Akce:*

- *klasické operátory z C (nebo Javy) – +,-,*,/,+=,-=,==,<=,>=*
- *příkazy – print, princ*
- *práce s proměnnými – deklarujeme přiřazením (s1 = $1, a = 5, hnuj = "fuj")*
- *jazyková konstrukce – if, for, while… (známe z C, nebo Javy)*

*code:*

```
# ypcat passwd | awk -F: '$3>=1000 && $3<=9999'
# ypcat passwd | awk 'END { print "Total users: " NR }'
# awk 'BEGIN { FS=":"; OFS=":" } { print $3,$1,$5 }' /etc/passwd
```

Příklady:

```
{ print "Číslo záznamu=" NR, "Počet položek=" NF, $0 }
```

**Takto zadaný program před kompletním záznamem vypíše číslo záznamu a počet položek v aktuálním záznamu.**

**Výstup můžeme rozdělit i do více výstupních souborů. Např. program:**

```
{ print $1 >"soubor1"; print $2 >"soubor2" }
```

**zapíše první položku do souboru `soubor1` a druhou položku do souboru `soubor2`. Lze použít i zápis >>. Potom se do souboru přidává za konec. Jméno souboru může být proměnná, obsah zapisovaný do souboru může být konstanta. Můžeme tedy napsat např.:**

```
{ print "nesmysl" >>$2 }
```

**Jako jméno souboru se použije obsah druhé položky (pozor, nesmí být prázdná). V tomto příkladě bude počet řádků v jednotlivých souborech znamenat četnost slov ve druhém poli.**

**Podobně lze výstup z akce print předat rourou procesu. Např. poslat poštou na adresu 'zaznamenej':**

```
{ print | "mail zaznamenej" }
```

***Poznámka:***

*awk vychází z C, spousta věcí, z něj se zde dá použít, nawk (pokročilejší verze awk) podporuje také výchozí funkce. Už i v awk si můžete napsat vlastní funkce.*

# Tahák na UOS 2008/2009

*sestavil Tomáš „Inza“ Jukin – http://www.dvojmo.cz*

## PŘÍKAZY – ADMINISTRACE

*chown, chmod, tar, unzip, exec, nice, renice, kill, nohup, ps, prstat, ptree, last*

### chown

*(CHange OWNer) mění vlastníka souboru*

`chown [-R] kdo[:skupina] co`

**Mění vlastníka souboru. Smí pouze superuživatel**

*Volby:*

- *-R rekurentní změna (včetně podadresářů)*

*code:*
```
# chown -R admin:staff /home/web/
```

### chmod

*(CHange MODe) mění přístupová oprávnění k souboru*

`chmod [-R] vzor cil`

**Mění přístupová práva k souboru v parametru cil. Nastavuje právě podle vzor. Vzor je ve formátu X+/-/=Y, kde X je jedno z a,u,g,o (All, User, Group, Others – všichni, vlastník, jeho skupina, ostatní) a Y je jedno z r,w,x (Read, Write, eXecute – čtení, zápis, spouštění).Operátor může být plus (přidá práva), mínus (odstraní práva), rovnítko (nastaví práva podle vzoru).**

*Volby:*

- *-R rekurentní změna (včetně podadresářů)*

*code:*
```
# chmod -R a+rw /home/web/
```

```
Př.: Udělá skript skript.sh spustitelným:
# chmod a+x skript.sh
```

***Poznámka:***

*Vzor může také být octalové číslo (např. 0777 pro plný přístup všem).Více o právech viz. man chmod*

### tar

*(Tape ARchiver) Serializer, spolupracuje s kompresory*

`tar [-c/t/x] [-vf] kam co`

**Komprese v UNIXu je realizována po částech. Zatímco na WIN je archivátor (RAR, nebo ZIP) program, který provádí dvě činnosti (serializaci a kompresi), na UNIXu je filozofie, že každý program dělá obvykle jen JEDNU věc, ale POŘÁDNĚ. Proto je i archivace rozdělena do dvou příkazů. Příkaz tar dělá serializaci (- z několika souborů udělá jeden). Vlastní kompresi (ale zase jenom tu kompresi – kompresi jednoho souboru) pak již provádějí jiné příkazy (zip, bzip2). Příkaz tar ale dnes již umí s těmito příkazy přímo interně spolupracovat**

*Volby:*

- *-t [Test archive] otestuje (zkontroluje, vypíše obsah) archivu*
- *-c [Create archive] vytvoří archiv*
- *-x [eXtract archive] rozbalí archiv*
- *-v ukecaný režim*
- *-f čtení z/zápis do souboru*
- *-z použij kompresi pomocí gzipu*
- *-j použij kompresi pomocí bzipu2*

*code:*
```
# tar -czvf mujarchiv.tar.gz mujadr_ke_kompresi
# tar -tzvf mujarchiv.tar.gz mujadr_k_otestovani
# tar -xzvf mujarchiv.tar.gz mujadr_k_rozbaleni
# tar -cf /dev/tape mujsoubor1 mujsoubor2
```

### unzip

*(UNZIP file) dekomprimuje soubory z formátu .ZIP*

`unzip co [kam]`

**Rozbalí .ZIP archiv zadaný parametrem *co* do umístění zadaného parametrem *kam***

*Volby:*

- *-x seznam souborů, které bude ignorovat*

*code:*
```
# unzip my.zip
# unzip my.zip /home/myfiles
```

***Poznámka:***

*Tento příkaz umí pouze rozbalovat .ZIP archivi, pro jejich kompresi slouží program „zip“. Na UNIXu ovšem není „.zippování“ nejlepším možným způsobem archivace. Vřele doporučuji GZip.*

### exec

*(EXECute) spustí program místo aktuální instance shellu exec*

**BASH normálně vytváří pro každý spuštěný program nový proces. Pomocí exec lze spustit program v aktuálním procesu shellu (místo něj). Tedy jím lze například změnit jeho vstup a výstup.**

*code:*
```
# exec >std.out 2>std.err
```

## nice

*(be NICE) spustí proces s nižší prioritou*

`nice [-X] [prikaz]`

**Spustí zadaný proces s nižší prioritou, lze ji uvést jako volbu „-s“, pokud není, použije se implicitní 10. Pokud zavoláme nice bez parametru, zjistíme prioritu aktuálního procesu**

*Volby:*

- *-X priorita, kde X je číslo od 1 do 20, čím vyšší, tím větší zpomalení (snížení priority)*

*code:*
```
# nice
# nice -7 sort velky_soubor > vystup.txt
# nice sort velky_soubor
```

***Poznámka:***

*Superuživatel může zadáním záporné priority prioritu i zvyšovat*

## renice

*(REset NICE) přenastaví prioritu již běžícímu procesu*

`renice +5 [-p pid | -u username]`

**Přenastaví prioritu již běžícímu procesu**

*Volby:*

- *-X kde X je číslo priority, stejné jak u nice*
- *-p PID cílového procesu*
- *-u uživatelské jméno, pak se stahuje na všechny procesy spuštěné zadaným uživatelem*

*code:*
```
# renice +5 -p 28734
```

## kill

*(KILL process) zasílá signály procesům*

`kill [-KILL] [-X | -name] PID`

**Pošle procesu signál. Je efektivní, jen když je zasláno vlastníkem procesu, nebo super uživatelem. Defaultně procesy ukončuje. Lze také zaslat jiný signál než k ukončení.**

*Volby:*

- *-KILL force kill, když nefunguje normální kill, tohleto zabere*
- *-X kde X je číslo signálu, který ze zašle místo ukončení*
- *-name kde name je název signál, který se zašle místo ukončení*
- *-l vypíše seznam dostupných signálů*

*code:*
```
# kill 12345
# kill -KILL 12345
```

## nohup

*(NO HangUPs) spouští programy nezávisle na aktuální relaci*

`nohup prikaz`

**Spustí zadaný příkaz nezávisle na aktuální relaci (tzn. že poběží i po skončení BASHe)**

*code:*
```
# nohup sort velky_soubor
```

## ps

*(Process Status) zobrazuje informace o procesech (aktuálního shellu)*

`ps [-U login]`

**Bez parametrů zobrazí informace o aktuálních procesech aktuálního shellu. S parametrem zobrazí procesy zadaného uživatele.**

*Volby:*

- *-U zobrazí procesy uživatele dle zadaného loginu (username)*

*code:*
```
# ps
# ps -U root
```

***Poznámka:***

*Příkaz má daleko více voleb viz man ps*

### prstat

*(PRocess STATistics) zobrazí seznam aktuálně běžících procesů (v rámci PC)*

```
prstat [-Z]
```

**Zobrazí real-time se obnovující informace o procesech v systému. Přepínačem -Z lze zobrazit shrnutí**

*Volby:*

- *-Z zobrazí i shrnutí*

*code:*

```
# prstat
# prstat -Z
```

***Poznámka:***

*Příkaz má mnohem více voleb, viz. man prstat*

### ptree

*(Process TREE) Zobrazí strom procesu se zadaným id*

```
Př.: Zobrazit strom aktuálního procesu:
# ptree $$
```

### last

*(LAST action) zobrazí seznam posledních akcí všech uživatelů*

```
last
```

**Zobrazí seznam (historii) všech provedených akcí všech uživatelů. Kvůli velikosti výstupu je vhodné zapsat do roury s less, či head.**

*code:*

```
# last | head
# last | less
```

***Poznámka:***

*Nejnovější jsou uvedeny nahoře*

## PŘÍKAZY – PROGRAMOVÉ STRUKTURY

*proměnné, pole, if, case, while, for, test, expr, let, shift, read, pomůcky pro ladění*

### Proměnné:

*Volby:*

- *$# Počet argumentu skriptu*
- *$0 Jméno skriptu*
- *$1, $2, … Argumenty skriptu*
- *$* = $1 $2 $3 ...*
- *$@ = $1 $2 $3 ...*
- *"$*" = "$1 $2 $3 ..."*
- *"$@" = "$1" "$2" "$3" ...*
- *$JMENO hodnota proměnné*
- *${JMENO} hodnota proměnné*
- *${JMENO:-text} je-li JMENO prázdné, pak vrátí text, jinak $JMENO*
- *${JMENO:=text} je-li JMENO prázdné, pak JMENO=text a vrátí $JMENO*
- *${JMENO:?text} je-li JMENO prázdné, pak vypíše text a konci (exit)*

*zrušení proměnné:*

```
unset JMENO
```

*vytvoření konstanty:*

```
JMENO=HODNOTA
readonly JMENO
```

### Pole:

*Přiřazeni:*

```
# JMENO[index]=HODNOTA
```

*Čtení:*

```
# ${JMENO[index]}
```

*Čtení všech položek:*

```
${JMENO[*]}
```

*Počet položek v poli:*

```
${#JMENO[*]}
```

### Příkaz if:

*Jednoduchá podmínka*

*code:*

```
#!/bin/sh
if [ $# -ne 1 ] ; then
  echo "volaní: $0 číslo_navratoveho_kodu"
  exit 2
fi
exit $1
```

## Příkaz case:

*Složená podmínka*

*code:*

```
#!/sbin/sh
case "$1" in
'start')
   [ -x /usr/lib/lpsched ] && /usr/lib/lpsched
;;
'stop')
   [ -x /usr/lib/lpshut ] && /usr/lib/lpshut
;;
*)
   echo "Usage: $0 { start | stop }"
   exit 1
;;
esac
```

## Příkaz while:

*Cyklus s neznámým počtem opakování*

*code:*

```
#!/bin/sh
MAX=5
I=1

while [ "$I" -le 10 ]
  do
    echo "Hodnota I je $I"
    I=`expr "$I" + 1`
  done
```

**Příklad:**

```
while :
do
        /bin/echo "Zadej cele číslo [0,...99][k=konec]: \c"
        read C
        case "$C" in
        k)
                break
        ;;
        [0-9]|[0-9][0-9] )
                echo "Druha mocnina čísla $C je `expr $C \* $C`."
        ;;
        *) echo "Špatný parametr."
        esac
done
```

## Příkaz for:

*Cyklus s pevným počtem opakování*

*code:*

```
#!/bin/sh
I=1

for E in Petr Jana Jiri Karel
  do
    echo "Element $I je $E."
    I=`expr $I + 1`
  done
```

## test, [ ... ]

*(TEST expression) testuje zadaný logický výraz*

*Volby:*

- *AND: vyraz1 -a vyraz2*
- *OR: vyraz1 -o vyraz2*
- *NOT: ! vyraz1*
- *\( přednostní vyhodnoceni \)*

*Operátory pro operace se SOUBORY:*

- *[ -f soubor ] # soubor existuje a je obyčejným souborem?*
- *[ -d soubor ] # soubor existuje a je adresářem?*
- *[ -s soubor ] # soubor existuje a Není prázdný?*
- *[ -e soubor ] # soubor existuje?*

- *[ -L soubor ] # soubor existuje a je symbolickým linkem?*
- *[ -r soubor ] # soubor existuje a má nastaveno právo r?*
- *[ -w soubor ] # soubor existuje a má nastaveno právo w?*
- *[ -x soubor ] # soubor existuje a má nastaveno právo x?*

Příklad:

- `# [ -r "$P" ] && echo "soubor $P je čitelný"`

*Přepínače operaci s RETEZCI:*

- *[ r1 = r2 ] Významy řetězce r1 a r2 jsou stejné?*
- *[ r1 != r2 ] řetězce r1 a r2 jsou různé?*
- *[ r1 \< r2 ] Je řetězec r1 v abeced před řetězcem r2?*
- *[ r1 \> r2 ] Je řetězec r1 v abeced za řetězcem r2?*
- *[ -z r1 ] Je řetězec r1 prázdný?*
- *[ -n r1 ] Není řetězec r1 prázdný?*

Příklady:

```
# A=Ales ; B=Jiri ; C="Dobry den"
# test "$B" \< "$C" ; echo $?
```

*Přepínače operaci s CISLY:*

- *[ n1 -eq n2 ] číslo n1 je rovno číslu n2?*
- *[ n1 -ne n2 ] číslo n1 Není rovno číslu n2?*
- *[ n1 -lt n2 ] číslo n1 je menší než číslo n2?*
- *[ n1 -gt n2 ] číslo n1 je větší než číslo n2?*
- *[ n1 -le n2 ] číslo n1 je menší nebo rovno číslu n2?*
- *[ n1 -ge n2 ] číslo n1 je větší nebo rovno číslu n2?*

Příklady:

```
# test 2 -lt 7 && echo "2 < 7"
```

## expr

*(EXPRession) počítá matematické výrazy*

*code:*

```
# N=`expr $N1 + 3`
# N=`expr $N1 - $N2`
# N=`expr 10 \* 21`
# N=`expr $N1 / $N2`
# N=`expr $N1 % 5`
# A=`expr \( 5 + 3 \) \* 2`;
```

## let nebo (( ))

*(LET it be) počítá matematické výrazy user-friendly. Není nutno používat $ pro volání proměnných*

*Příklady:*

- *((N = N1 + 3))*
- *((N = N1 - N2))*
- *((N = 10 * 21))*
- *((N = N1 / N2))*
- *((N = N1 % 5))*
- *((N=2#1011)) #zaklad soustavy*
- *((N= 2#1011 << 3)) #bitový posun doleva*
- *((N= 2#1011 >> 3)) #bitový posun doprava*

## shift

*(SHIFT it) provede posun hodnot parametru*

*Vlastnosti:*

- *posune hodnoty parametru vlevo: $i = ${i+n}*
- *odebere Parametry z $* a $@*
- *dekrementuje: $# = $# - n*

Příklad:

```
#!/bin/bash
I=1
echo "Počet parametru: $#"

while [ $# -gt 0 ]
do
        echo "Hodnota parametru $I: $1"
        shift
        I=`expr $I + 1`
done
```

## read

*(READ from input) Čte ze standardního vstupu*

*Použití:*

```
# read P1 P2 P3
```

*Popis:*

*Přečte jednu řádku ze vstupu. Podle proměnné $IFS rozdělí načtenou řádku na jednotlivé hodnoty.*
*Uloží první hodnotu do proměnné P1, druhou položku do proměnné P2 a ostatní hodnoty do proměnné P3.*

Příklad:

```
#!/bin/sh
while :
do
  /bin/echo "Zadej cele číslo [0,...99][k=konec]: \c"
  read C
  case $C in
    k)
       break
       ;;
    [0-9]|[0-9][0-9] )
       echo "Druha mocnina čísla $C je `expr $C \* $C`."
       ;;
    *) echo "Špatný parametr."
  esac
done
```

Př.: Čtení dat ze souboru:

```
#!/bin/sh
echo "Informace od uživatelích v /etc/passwd"
IFS=":"
while read JMENO NIC UID GID POPIS DIR LOGSHELL
do
  echo "Účet $JMENO má:"
  echo " UID=$UID"
  echo " GID=$GID"
  echo " HOME=$DIR"
  echo " SHELL=${LOGSHELL:-Není definován}"
done < /etc/passwd
```

### Pomůcky pro ladění:

*jak efektivně ladit skripty?*

*code:*

```
sh -v ./script # předem echuje Příkazy
sh -x ./script # předem echuje Příkazy, nahrazené spec. znaky
```

## OSTATNÍ

*To, co se nevešlo jinam…*

**Existují příkazy „ypcat“, atd. Jsou shodné se svými protějšky bez „yp“, akorát že data načítají z NISu (síťového úložiště)**

Př.: Porovnání slov z prvních 300 řádků výstupu man man, které nejsou v souboru /usr/share/lib/dict/words:

```
man man \
| head -300 \
| tr '[A-Z]' '[a-z]' \
| tr -cs '[a-z]' '[\n*]' \
| sort \
| uniq \
| comm -23 - /usr/share/lib/dict/words \
| tee unknown.words \
| wc -l
```