Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

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

- *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.

- *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“.

Všechny výše uvedené způsoby lze vzájemně kombinovat:

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)*

PŘÍKAZY – PRÁCE S PROSTŘEDÍM

type, which, who, whoami, whereis, date

type

(command TYPE) vyhledá spustitelný soubor skriptu v shellové cestě

```
type param1 [param2 […]]
```

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

code:
```
# type man
# type which
```

which

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

`which param1 [param2 […]]`

code:
```
# which who
```

who

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

`who`

code:
```
# who
```

whoami

(WHO AM I) vypíše přihlašovací jméno aktuálního uživatele

`whoami`

code:
```
# whoami
```

whereis

(WHERE IS this) zkouší nalézt dané soubory pomocí databáze

`whereis param1`

code:
```
# whereis who
```

date

(show DATE) vypíše aktuální datum a čas

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

Volby:

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

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

PŘÍKAZY – NAVIGACE

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

cd

(Change Directory) změní aktuální adresář

`cd [param1]`

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

ls

(LiSt directory) zobrazí obsah adresáře

`ls [-alFLhi] [param1 [param2 […]]]`

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`

code:
```
# pwd
```

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

mkdir

(MaKe DIRectory)vytvoří adresář

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

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`

Volby:

- *-p maže i s cestou*

code:

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

cp

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

`cp [-Rif] co 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
```

mv

(MoVe file) přejmenovává/přesunuje soubory

`mv [-if] co kam`

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
```

rm

(ReMove file) odstraňuje soubory

`rm [-Rif] 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
```

ln

(create LiNk) vytvoří odkaz (link)

`ln [-s] cil odkaz`

Volby:

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

code:

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

PŘÍKAZY – ZÁKLADNÍ FILTRY

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

cat

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

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

`cat [muj_soubor]`

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`

Volby:

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

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

head

(HEAD of file) zobrazuje řádky od začátku souboru

```
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]
```

head + tail

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

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]`

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]`

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

Volby:

- *-m upovídaný prompt*
- *-N číslování řádků*
- *-s „squezee“ 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
```

more

(show MORE) prohlížeč dlouhých souborů

more

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,..]

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
```

uniq

(make it UNIQue) odstraňuje duplicity ze setříděného vstupu

uniq [*soubor*]

code:
```
# ls | sort | uniq
```

tee

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

tee [*soubor*]

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

tr

(TRanslate) překládá znaky

tr [-s] *co cim*

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
```

grep

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

grep [-v] [-l] *regexp*

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*

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

- *[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'
```

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

egrep

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

Vlastnosti:

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

cmp

(CoMPare files) porovnává soubory binárně

cmp *soubor1 soubor2*

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]

Volby:

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

code:
```
# comm -12 file1 file2
```

diff

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

diff *souborA souborB*

Volby:

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

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

patch

(PATCH file) aplikuje záplaty, vytvořené programem diff

patch < *záplata*

Volby:

- *-R revertuje (vezme zpět) záplatu*

code:
```
# patch < fix.txt
```

find

(FIND file) hledá soubory

Hledá soubory a adresáře podle specifikovaných vlastností

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

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

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`

`'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
```

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*

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) – +,-,*,/,+=,-=,==,<=,>=*

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

- *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
```

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`

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`

Volby:

- *-R rekurentní změna (včetně podadresářů)*

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

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

tar

(Tape ARchiver) Serializer, spolupracuje s kompresory

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

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]`

Volby:

- *-x seznam souborů, které bude ignorovat*

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

exec

(EXECute) spustí program místo aktuální instance shellu exec

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

nice

(be NICE) spustí proces s nižší prioritou

`nice [-X] [prikaz]`

Volby:

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

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

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

renice

(REset NICE) přenastaví prioritu již běžícímu procesu

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

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`

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`

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

ps

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

`ps [-U login]`

Volby:

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

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

prstat

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

`prstat [-Z]`

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`

code:
```
# last | head
# last | less
```

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

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
```

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

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?*

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?*

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? expr*

(EXPRession) počítá matematické výrazy

code:

```
# N=`expr $N1 + 3`
```

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*

read

(READ from input) Čte ze standardního vstupu

Použití:

```
# read P1 P2 P3
```

Tahák na UOS 2008/2009 – Přehled (zkrácená verze)

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

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.

Pomůcky pro ladění:

jak efektivně ladit skripty?

code:

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

POZNÁMKA

Toto je pouze neúplný, kratší tahák pro ty, co se vyznají. Úplný tahák lze stáhnout z mého webu na http://beta.dvojmo.cz/fel/y36uos/ …