Skripty a funkce

SKRIPTY	
true	vrátí úspěšný (nulový) návratový kód
false	vrátí neúspěšný (nenulový) návratový kód
test <podminka> | **[** <podminka> **]** | **[[** <podminka> **]]**	vyhodnotí danou podmínku a v závislosti na jejím splnění vrátí úspěšný či neúspěšný návratový kód; k testování existence specifických souborů slouží následující volby, jejichž argumentem je název souboru: **-e** jakýkoliv soubor, **-f** běžný soubor, **-b** blokové zařízení, **-c** znakové zařízení, **-d** adresář, **-h** symbolický odkaz, **-s** soubor s nenulovou velikostí, **-r** soubor s právem ke čtení, **-w** soubor s právem k zápisu, **-x** soubor s právem ke spuštění, **-u** soubor s SUID bitem, **-g** soubor s SGID bitem, **-k** soubor se sticky bitem, **-p** pojmenovaná roura, <soubor1> **-nt** <soubor2> soubor1 je novější než soubor2, <soubor1> **-ot** <soubor2> soubor1 je starší než soubor2; k testování celých čísel slouží operátory: **-eq** je rovno, **-ne** není rovno, **-lt** menší než, **-le** menší nebo rovno, **-gt** větší než, **-ge** větší nebo rovno; k testování řetězců: **=** je rovno, **!=** není rovno, **-z** <retezec> délka řetězce je nulová, **-n** <retezec> délka řetězce je nenulová; rozšířené testování umožňují operátory: **-a** logické A (oba výrazy jsou pravdivé), **-o** logické NEBO (alespoň jeden výraz je pravdivý), **!** logické NE (výraz je nepravdivý); při použití **[[ ]]** lze navíc testovat výrazy jako <retezec> **=** / **==** <vzor> (řetězec obsahuje vzor), <retezec> **!=** <vzor> (řetězec neobsahuje vzor), <retezec> **=~** <vzor> (řetězec odpovídá regulárnímu výrazu), **!** <retezec> **=~** <vzor> (řetězec neodpovídá regulárnímu výrazu), <retezec1> **<** <retezec2> nebo <retezec1> **>** <retezec2> (řetězec1 podle abecedy předchází nebo následuje řetězec2) a využít **&&** pro vyhodnocení druhého výrazu pokud první skončil úspěšně, **||** pro vyhodnocení druhého výrazu pokud první skončil neúspěšně a **()** pro seskupování výrazů `$ test -f *.txt	$ [-f *.txt]` (návratový kód je 0, existuje-li právě 1 soubor s příponou „.txt“) `$ [[-n $(find . -name "*.txt")]]` (návratový kód je 0, existuje-li 1 a více souborů s příponou „.txt“) `$ [! -d PROD]` (návratový kód je 0, pokud adresář neexistuje) `$ [[abc == *c]]` (návratový kód je 0, obsahuje-li řetězec „abc“ vzor „*c“) `$ [["abc" =~ .*c]]` (návratový kód je 0, odpovídá-li řetězec „abc“ regulárnímu výrazu „.*c“) `$ [[a < z]]` (návratový kód je 0, předchází-li řetězec „a“ v abecedním pořadí řetězec „z“) `$ [[$(awk -F ":" '/^root/ {print $4}' /etc/passwd) -eq 0]]` (návratový kód je 0, je-li GID uživatele root 0) `$ [$? -ne 0] && echo $?` (vypíše návratový kód předchozího příkazu, skončil-li neúspěšně) `$ [$PWD = $HOME] && echo "doma" \|\| pwd` (vypíše „doma“, nachází-li se přihlášený uživatel v domovském adresáři, jinak zobrazí cestu k pracovnímu adresáři) `# duid=$(awk -F ":" '{print $3}' /etc/passwd \| sort \| uniq -d); [[-z "$duid"]] && echo "NONE" || echo "$duid"` (vypíše duplicitní UID v systému)
if <seznam>**; then** <seznam>**;** [**elif** <seznam>**; then** <seznam>**;**] ... [**else** <seznam>**;**] **fi**	**if** uvádí podmínku, je-li pravdou, pak **then** určí požadovanou akci, není-li pravdou, **elif** stanoví další podmínku a **then** určí další akci, jinak **else** určí akci alternativní, seznam podmínek ukončuje **fi** `#!/bin/bash` `# Skript kontroluje, zda je přihlášený uživatel root. Pokud je uživatel root, vyzve k zadání příkazů. Pokud uživatel není root, informuje jej, že nemá povoleno spouštět skript a ukončí se s chybovou zprávou.` `if [[ $EUID -ne 0 ]]; then` `  echo "Uzivatel '$(whoami)' neni opravnen spustit ${0##*/}."` `  exit 1` `fi` `read -p "Zadejte sve prikazy: " prikazy` `eval "$prikazy"` `echo "Skript byl uspesne dokoncen."` `#!/bin/bash` `# Skript ověří, zda je zadaný balíček nainstalován.` `echo -n "Zadejte nazev balicku: "` `while read PACKAGE` `do` `  rpm -q $PACKAGE > /dev/null` `  if [ $? -eq 0 ]` `  then` `  echo "ANO, $PACKAGE je nainstalovan."` `  else` `  echo "NE, $PACKAGE neni nainstalovan."` `fi` `. $0` `done`

SKRIPTY	
case <slovo> **in** <vzor>[|<vzor>]...**)** <seznam>**;; esac**	vykoná příkaz na základě shody slova se vzorem; slovo představuje libovolnou proměnnou, vzor „*“ uvádí akci, pokud shoda nenastane, seznam vzorů ukončuje „)“ a skupina příkazů se zakončí „;;“ `#!/bin/bash` `# Skript podle operace nainstaluje, aktualizuje či odinstaluje daný balíček.` `echo -n 'Zadejte operaci a balicek: '` `read OPERACE BALICEK` `case $OPERACE in` `  nainstaluj)  yum install -y $BALICEK;;` `  aktualizuj)  yum update -y $BALICEK;;` `  odinstaluj|smaz) yum remove -y $BALICEK;;` `  *)  echo 'Neznama operace.'` `  echo 'Pouziti: {nainstaluj|aktualizuj|odinstaluj|smaz} <balicek>';;` `esac`
read <promenna ...>	čte řádek ze STDIN, rozdělí ho na slova a ta přiřadí danému počtu proměnných v odpovídajícím pořadí pro další zpracování; pokud je více slov než proměnných, zbývající slova a jejich oddělovače jsou přiřazeny poslední proměnné, pokud je proměnných více než slov, zbývajícím proměnným jsou přiřazeny prázdné hodnoty, **-a** <indexovane_pole> určí indexované pole, kde jsou slova přiřazena k sekvenčním indexům, **-n** <n> čte pouze daný počet znaků, **-p** <vyzva> určí výzvu, která se uživateli zobrazí před čtením vstupu, **-r** ruší znaku „\“ speciální význam, **-s** nezobrazí žádné znaky na vstupu, **-t** <n> nastaví časový interval v sekundách pro čtení vstupu `$ cat cisla.txt | { read a; read b; read c; echo "$a/$b/$c";}` (čte první tři řádky souboru, které vypíše na jeden řádek za sebou oddělené lomítkem) `#!/bin/bash` `# Skript vyzve uživatele k zadání jména a hesla a potvrdí jeho přijetí.` `echo -n "Jmeno: "; read jmeno` `echo -n "Heslo: "; read -s heslo` `echo` `echo "Heslo pro $jmeno zadano."`
while <seznam1>**; do** <seznam2>**; done**	spouští opakovaně „seznam2“, dokud „seznam1“ vrací úspěšný návratový kód `#!/bin/bash` `# Skript otvírá čtyři terminálová okna.` `i=0` `while [ $i -lt 4 ]; do xterm &` `  i=$[$i+1]; done` `#!/bin/bash` `# Skript vytváří soubor s informacemi o právě přihlášených uživatelích v intervalu pěti minut.` `while true; do` `  who > info-$(date | awk '{print $2,$3,$4,$6}' | sed 's/ /_/g').txt` `  sleep 300; done` `#!/bin/bash` `# Skript vyzve uživatele k zadání jména vzdáleného počítače, na kterém spustí příkaz, který zjistí verzi distribuce Linuxu a výsledek uloží do souboru na lokálním počítači.` `echo -n "Jmeno pocitace: "; while read hostname; do ssh $hostname 'echo "Distribuce Linuxu": $(cat /etc/redhat-release)' > ${hostname}_info && echo -n "Jmeno pocitace: "; done`

SKRIPTY	
until <seznam1>**; do** <seznam2>**; done**	spouští opakovaně „seznam2“, dokud „seznam1“ vrací neúspěšný návratový kód `#!/bin/bash` `# Skript vypíše čísla od 0 do 99.` `a=-1` `until [ $a -eq 99 ]; do` `  a=$(expr $a + 1)` `  echo $a; done` `#!/bin/bash` `# Skript kopíruje soubory z domovského do webového adresáře, kde každou hodinu vytvoří nový adresář, přičemž v případě jeho zaplnění na více než 90 % smaže adresáře starší 30 dní.` `while true; do` `  DISKFUL=$(df -h $WEBDIR | grep "^/" | \` `  awk '{print $5}' | cut -d "%" -f1 -)` `  until [ $DISKFUL -ge 90 ]; do` `  WEBDIR=/var/www/webcam` `  DATE=$(date +%Y%m%d)` `  HOUR=$(date +%H)` `  mkdir $WEBDIR/"$DATE"` `  while [ $HOUR -ne 00 ]; do` `  PICDIR=/home/user/pics` `  DESTDIR=$WEBDIR/"$DATE"/"$HOUR"` `  mkdir "$DESTDIR"` `  cp $PICDIR/*.jpg "$DESTDIR"/` `  sleep 3600` `  HOUR=$(date +%H); done` `  DISKFULL=$(df -h $WEBDIR | grep "^/" | \` `  awk '{print $5}' \| cut -d "%" -f1 -)` `  done` `TOREMOVE=$(find $WEBDIR -type d -mtime +30)` `  for i in $TOREMOVE; do` `  rm -rf "$i"; done` `done`

SKRIPTY	
for <promenna> [**in** <slovo> ...**;**] **do** <seznam>**; done**	přiřadí proměnné postupně hodnotu všech slov nebo všech pozičních parametrů skriptu a provede seznam příkazů pro každou z položek `#!/bin/bash` `# Skript zobrazí každý argument příkazového řádku spolu s jeho odpovídajícím číslem.` `count=1` `for arg; do` `  echo "Argument $count: $arg"` `  ((count++))` `done` `#!/bin/bash` `# Skript vytvoří uživatele uvedené v souboru „uzivatele“ v pracovním adresáři, přidá je do skupiny „admins“ a nastaví heslo „password“.` `for uzivatel in $(cat uzivatele); do` `  useradd -g admins $uzivatel` `  echo password | passwd --stdin $uzivatel` `done` `#!/bin/bash` `# Skript vypíše obsah pracovního adresáře.` `for x in $(ls -F)` `do` `  echo "V adresari $(pwd) se nachazi soubor ci adresar: $x"` `done` `#!/bin/bash` `# Skript spočítá počet položek v pracovním adresáři.` `POC=0` `for nazev in *` `  do` `  POC=$(($POC+1))` `  done` `echo "V adresari $(pwd) je $POC polozek."` `#!/bin/bash` `# Skript změní formát všech souborů .tif na .jpg.` `for pic in *.tif; do` `  convert "$pic" "$(echo "$pic" | sed 's/\.tif/.jpg/')"` `done` `#!/bin/bash` `# Skript zjistí, které z uvedených procesů běží, vypíše jejich jména včetně počtu řádků, které zabírají a nakonec zobrazí výpis všech běžících procesů a celkový počet řádků.` `services="httpd pmon lsnr sap"` `for service in $services` `  do running_processes="$(ps -ef | grep $service \| egrep -v 'grep\|vnc\|client\|API')"` `  if [ -n "$running_processes" ]` `  then echo` `  echo $service:` `  echo` `  echo "$running_processes"` `  echo` `  echo "radku: $(echo "$running_processes" | wc -l)"` `  echo` `  echo "=========="` `  fi` `  done` `echo` `echo "Souhrn:"` `echo` `ps -ef` `echo` `echo "radku: $(ps -ef | wc -l)"` `#!/bin/bash` `# Skript se připojí ke všem serverům, jejichž IP adresy jsou uvedeny v souboru „servers“, na nich spustí příkazy, které zjistí jméno serveru a všechny jeho IP adresy a výsledek zapíše do souboru v lokálním počítači.` `for ip in $(< servers); do ssh $ip 'hostname; ifconfig | awk '/inet / {print $2}' \| grep -v 127.0.0.1' >> servers_info.txt; done` `#!/bin/bash` `# Skript se připojí ke vzdáleným počítačům, na kterých nainstaluje software pro danou službu, kterou poté nastartuje a ověří její status.` `for x in {a..c}; do echo "=== node${x} ==="; ssh node${x} 'yum install -y device-mapper-multipath; systemctl enable --now multipathd; systemctl status multipathd'; echo; done`

SKRIPTY	
select <promenna> [**in** <slovo> ...**;**] **do** <seznam>**; done**	přiřadí proměnné postupně hodnotu všech slov nebo všech pozičních parametrů skriptu, jejichž očíslovaný seznam zobrazí společně s výzvou pro uživatele k zadání čísla položky a poté provede seznam příkazů pro vybranou položku (interaktivní obdoba příkazu „for“); přečtený řádek se uloží do proměnné „REPLY“

```
#!/bin/bash
# Skript zobrazí menu s nabídkou služeb. Po vybrání jedné z nich se objeví vnořené menu s nabídkou operací, které lze provést.

while true; do
PS3="Vyberte sluzbu nebo odejdete: "
select svc in cups httpd sshd odejit
do
  case $svc in
    cups)
      PS3="Vyberte operaci nebo odejdete: "
      select opt in start stop restart status odejit
      do
        case $opt in
          start)
            systemctl start cups
            break;;
          stop)
            systemctl stop cups
            break;;
          restart)
            systemctl restart cups
            break;;
          status)
            systemctl status cups
            break;;
          odejit)
            break;;
          *)
            echo "Neplatna volba $REPLY"
            break;;
        esac
      done
      break;;
    httpd)
      PS3="Vyberte operaci nebo odejdete: "
      select opt in start stop restart status odejit
      do
        case $opt in
          start)
            systemctl start httpd
            break;;
          stop)
            systemctl stop httpd
            break;;
          restart)
            systemctl restart httpd
            break;;
          status)
            systemctl status httpd
            break;;
          odejit)
            break;;
          *)
            echo "Neplatna volba $REPLY"
            break;;
        esac
      done
      break;;
    sshd)
      PS3="Vyberte operaci nebo odejdete: "
      select opt in start stop restart status odejit
      do
        case $opt in
          start)
            systemctl start sshd
            break;;
          stop)
            systemctl stop sshd
            break;;
          restart)
            systemctl restart sshd
            break;;
          status)
            systemctl status sshd
            break;;
          odejit)
            break;;
          *)
            echo "Neplatna volba $REPLY"
            break;;
        esac
      done
      break;;
    odejit)
      break 2;;
    *)
      echo "Neplatna volba $REPLY";;
  esac
done; done
```

SKRIPTY	
break [<n>]	ukončí definitivně daný cyklus ve smyčce „for“, „while“, „until“, nebo „select“ na základě určitých kritérií, *n* určí počet úrovní vnořených cyklů `#!/bin/bash` `# Skript spustí smyčku, která by měla procházet cykly od 1 do 5, ale předčasně ji ukončí během třetího cyklu, když hodnota proměnné „cyklus“ dosáhne 3.` `for cyklus in {1..5}; do` `  if [ "$cyklus" -eq 3 ]; then` `  echo "Prerusuji behem cyklu $cyklus"` `  break` `  fi` `  echo "Cyklus: $cyklus"` `done` `echo "Smycka skoncila."`
continue [<n>]	přeskočí daný cyklus ve smyčce „for“, „while“, „until“, nebo „select“ na základě určitých kritérií a pokračuje v dalším, *n* určí úroveň cyklu `#!/bin/bash` `# Skript spustí smyčku, která by měla procházet cykly od 1 do 5, ale přeskočí třetí cyklus, když hodnota proměnné „cyklus“ dosáhne 3 a pokračuje v dalším cyklu.` `for cyklus in {1..5}; do` `  if [ "$cyklus" -eq 3 ]; then` `  echo "Preskakuji cyklus $cyklus"` `  continue` `  fi` `  echo "Cyklus: $cyklus"` `done` `#!/bin/bash` `# Skript převádí velká písmena v názvech souborů na malá.` `LIST=$(ls)` `for nazev in $LIST; do` `  if [[ $nazev != *[[:upper:]]* ]]; then` `  continue; fi` `ORIG=$nazev` `NEW=$(echo $nazev | tr 'A-Z' 'a-z')` `mv $ORIG $NEW` `echo "Novy nazev pro $ORIG je $NEW"` `done`
getopts <seznam_voleb> <jmeno> [<argumenty>]	rozdělí (zpracuje) poziční parametry (volby a argumenty), které jsou předány shellovému skriptu; <seznam_voleb> určí seznam podporovaných znaků voleb, následuje-li za znakem dvojtečka (:), očekává se, že volba bude mít argument, je-li dvojtečka před prvním znakem, potlačí se hlášení chyb; při každém volání „getopts“ získá další volbu z pozičních parametrů a umístí znak volby do proměnné <jmeno>, pokud volba neodpovídá těm, které jsou definovány v <seznam_voleb>, „getopts“ nastaví proměnnou <jmeno> na otazník (?) a pokud je potlačeno hlášení chyb, nalezený znak volby se umístí do proměnné „OPTARG“; vyžaduje-li volba argument, „getopts“ umístí tento argument do proměnné „OPTARG“, pokud není nalezen požadovaný argument a je potlačeno hlášení chyb, umístí se dvojtečka do proměnné <jmeno> a „OPTARG“ se nastaví na nalezený znak volby; při spuštění skriptu musí volby, které vyžadují argument, následovat jejich argumenty, než se použije další volba `#!/bin/bash` `# Skript povolí pro své spuštění používat pouze přesně definované volby a očekávané argumenty.` `pouziti() {` `  echo "Pouziti: $0 {-x\|-y <retezec>}"` `  exit 1` `}` `[[ ! "$@" =~ -.+ ]] && pouziti` `while getopts :xy: volba` `do` `  case $volba in` `  x)` `  echo "Je pouzita volba '-x'."` `  ;;` `  y)` `  echo "Je pouzita volba '-y' s argumentem '$OPTARG'."` `  ;;` `  \?)` `  echo "Neplatna volba '-$OPTARG'."` `  pouziti` `  ;;` `  :)` `  echo "Volba '-$OPTARG' vyzaduje argument."` `  pouziti` `  ;;` `  esac` `done`
exit [<n>]	ukončí skript s daným návratovým kódem, není-li uveden, návratovým kódem skriptu je návratový kód posledního příkazu spuštěného uvnitř skriptu

FUNKCE	
function <funkce> **{** <seznam>**;}** | <funkce> **()** **{** <seznam>**;}**	definuje funkci, jejíž obsah je uzavřen do složených závorek; funkce se provádí v běžném shellu a využívá se zejména v případech, kdy je třeba provést danou operaci (posloupnost příkazů) opakovaně; spustí se zadáním jejího názvu a standardně vrací návratový kód posledního provedeného příkazu, není-li stanoveno jinak příkazem „return“; uvnitř funkce lze příkazem „local“ definovat lokální proměnnou (ta se dědí i do vnořených funkcí), pokud existuje globální proměnná stejného názvu, je ve funkci potlačena; funkci lze používat trvale (i po dalším spuštění shellu) jejím uvedením v *~/.bashrc*; informace o argumentech předaných skriptům či funkcím jsou uloženy ve zvláštních proměnných („$#“ = celkový počet argumentů, „$<n>“ = daný argument v pořadí v rozmezí 1–9, „${<n>}“ = daný argument v pořadí v libovolném rozmezí, „$*“ či „$@“ = všechny argumenty) `# Funkce „test“ vytvoří soubor, který je zadán jako argument a nastaví mu příslušná přístupová práva. Pokud soubor stejného jména již existuje nebo zadaný název obsahuje pouze mezeru či zcela chybí, vypíše se chybové hlášení.` `function test` `{` `if ! [[ -e $* || $* = '' ]]; then` `  touch $*; chmod 755 $*` `else` `  echo "Chyba"` `fi` `echo "Počet argumentů: $#"` `echo "Všechny argumenty: $*"` `echo "Druhý argument: $2"` `}` `# Funkce „uzivatel“ zjistí, zda uživatel, který je zadán jako argument, existuje v systému, a pokud ano, vypíše o něm podrobné informace.` `uzivatel ()` `{` `if [ $(grep -ic ^$1 /etc/passwd) -eq 0 ]; then` `  echo "Uzivatel $1 neexistuje."` `else` `  echo "Uzivatel $1 existuje." && finger $1` `fi` `}` `# Funkce „procento“ vypočítá procentuální hodnotu druhého číselného argumentu z prvního, přičemž první argument představuje hodnotu 100 %.` `procento () {` `local x=$1 y=$2` `printf "$x = 100 %%\n$y = x %%\n"` `echo "$y = $[$(( 100 * $y )) / $x] % z $x"` `}` `# Funkce „pouziti“ vypíše možné volby pro spuštění skriptu, pokud nebyla žádná platná volba použita, vrátí návratový kód 1.` `pouziti () { echo "Pouziti: $0 {-start|-stop|-restart}";` `exit 1;}` `# Funkce „velikost“ vypočítá velikost zadaného adresáře.` `velikost() {` `if [ -z "$1" ]; then echo "Pouziti: velikost <adresar>";` `return 1; fi` `[ -d "$1" ] || { echo "Adresar nenalezen: $1"; return 1; }` `echo "Skenuji adresar: $1 ..."` `find "$1" -type f -printf '%s\n' | \` `awk '{sum += $1} END {printf "Bytes: %d | MB: %.2f | GB: %.2f\n", sum, sum/1024/1024, sum/1024/1024/1024}'` `}`

FUNKCE	
local [<promenna>[=<hodnota>] ...]	definuje lokální proměnnou funkce, **-a** určí indexované pole, **-A** určí asociativní pole; bez argumentu zobrazí všechny lokální proměnné funkce
return [<n>]	ukončí funkci s daným návratovým kódem, není-li uveden, návratovým kódem funkce je návratový kód posledního příkazu spuštěného uvnitř funkce