25. Shell a programování v shellu

Obsah

- Druhy shellů a jejich základní funkce
- Metaznaky a znaky zvláštního významu v shellu
- Aliasy
- Proměnné
- Poziční parametry
- Pojem skript, způsoby spouštění skriptu, ladění skriptu
- Základní programové konstrukce (podmíněné příkazy, cykly)

Druhy shellů a jejich základní funkce

Shell je program, který poskytuje rozhraní mezi uživatelem a jádrem (kernelem) operačního systému. V unixových systémech slouží shell jako příkazový interpret, který: - Čte příkazy z terminálu nebo ze skriptů - Interpretuje jejich význam - Spouští je - Vrací výsledky

Hlavní rodiny shellů

1. Bourne Shell Family

- **Bourne Shell (sh)** - původní shell vyvinutý Stephenem Bournem v Bellových laboratořích
- **Bourne-Again Shell (bash)** - vylepšená verze sh, nejrozšířenější shell v linuxových distribucích
- **Korn Shell (ksh)** - zpětně kompatibilní s sh, přidává mnoho nových funkcí
- **Z Shell (zsh)** - kombinuje vlastnosti bash, ksh s mnoha vylepšeními

2. C Shell Family

- **C Shell (csh)** - vyvinutý na univerzitě Berkeley s podobnou syntaxí jako jazyk C
- **TENEX C Shell (tcsh)** - vylepšená verze csh s lepší interaktivitou

BASH (Bourne-Again Shell)

BASH je dnes nejpoužívanější shell v Linuxu, který přináší mnoho vylepšení oproti původnímu Bourne Shellu: - Historie příkazů - Doplňování tabulátorem - Manipulace s jobsy - Aliasy a funkce - Aritmetické operace - Pokročilé programování skriptů

Struktura příkazů v BASH

`příkaz [přepínače] [argumenty]`

Příklad:

```
ls -la /home
```

- `ls` je příkaz
- `-la` jsou přepínače (kombinace `-l` a `-a`)
- `/home` je argument

Oddělovače příkazů

- **Mezera** - odděluje příkaz, přepínače a argumenty
- **Tabulátor** - funguje stejně jako mezera
- **Středník (`;`)** - umožňuje zapsat více příkazů na jeden řádek

  ```
  echo "První příkaz"; echo "Druhý příkaz"
  ```

- **Nový řádek** - odděluje příkazy

Úrovně Unixu

Úroveň	Příklad
Aplikace	Grafické uživatelské rozhraní (GUI)
Aplikace	Příkazové rozhraní (CLI) - shell
Jádro	Obecné funkce (správa paměti, systémová volání)
Jádro	Ovladače zařízení (hardware specific)
Hardware	Fyzické komponenty počítače

Typy příkazů v shellu

Vnitřní příkazy

- Jsou implementovány přímo v shellu
- Nevytváří se nový proces při spuštění
- Příklady: `cd`, `echo`, `alias`, `export`, operace s proměnnými

Vnější příkazy

- Jsou to samostatné programy mimo shell
- Při spuštění se vytváří nový proces
- Shell je hledá v cestách definovaných v proměnné `$PATH`
- Příklady: `ls`, `cat`, `grep`, `mkdir`

Pro zjištění, zda je příkaz vnitřní nebo vnější, můžete použít příkaz `type`:

```
type cd     # cd is a shell builtin
type ls     # ls is /bin/ls
```

Metaznaky a znaky zvláštního významu v shellu

Metaznaky jsou speciální znaky, které shell interpretuje speciálním způsobem.

Základní metaznaky

Metaznak	Význam
`~`	Domovský adresář uživatele (např. `~user` je domovský adresář uživatele "user")
`;`	Odděluje příkazy na jednom řádku
`\`	Escape znak - ruší speciální význam následujícího znaku
`#`	Označuje komentář (zbytek řádku je ignorován)
`|`	Roura (pipe) - přesměruje výstup jednoho příkazu na vstup druhého
`>`	Přesměrování výstupu do souboru (přepíše soubor)
`>>`	Přesměrování výstupu - připojí na konec souboru
`<`	Přesměrování vstupu ze souboru
`&`	Spuštění příkazu na pozadí, nebo spojení deskriptorů
`$`	Indikuje, že jde o proměnnou
`` ` `` nebo `$()`	Substituční výrazy - provede příkaz a vloží jeho výstup

Zástupné znaky (wildcards) pro práci se soubory

Zástupný znak	Význam
`*`	Libovolný počet libovolných znaků (kromě `.` na začátku jména)
`?`	Právě jeden libovolný znak (kromě `.` na začátku jména)
`[abc]`	Jeden znak z uvedené množiny (a, b nebo c)
`[a-z]`	Jeden znak z uvedeného rozsahu (a až z)
`[!abc]`	Jeden znak, který není v uvedené množině

rozdíl mezi zástupnými znaky a regulárními výrazy je, že zástupné znaky se aplikují na soubory ale regulérní výrazy pracují s textem

Příklady:

```
# Všechny soubory s příponou .txt
ls *.txt

# Všechny soubory začínající písmenem 'a' nebo 'b' s libovolným jedním znakem
ls [ab]?.txt

# Všechny soubory začínající na číslici
ls [0-9]*
```

Ochrana (escape) znaků

Pro zabránění speciální interpretace metaznaků shellem můžete použít několik metod:

Zpětné lomítko `\` Ruší speciální význam následujícího znaku.

```
echo Cena: \$10  # Vypíše: Cena: $10
```

Jednoduché uvozovky `'...'` Chrání všechny znaky mezi nimi. Uvnitř jednoduchých uvozovek si žádný znak nezachovává svůj speciální význam.

```
echo 'Proměnná $HOME obsahuje znak *'  # Vypíše doslovně, nerozvine $HOME ani *
```

Dvojité uvozovky `"..."` Chrání většinu speciálních znaků, ale umožňují náhradu proměnných a příkazovou substituci.

```
echo "Váš domovský adresář je $HOME"  # Nahradí $HOME skutečnou hodnotou
```

Aliasy

Aliasy umožňují definovat vlastní zkratky nebo alternativní názvy pro příkazy.

Vytvoření aliasu

```
alias ls='ls --color=auto'  # Barevný výpis ls
alias ll='ls -la'           # Dlouhý výpis včetně skrytých souborů
alias c='clear'             # Zkratka pro vymazání obrazovky
```

Zobrazení dostupných aliasů

```
alias  # Zobrazí všechny definované aliasy
```

Odstranění aliasu

```
unalias c  # Odstraní alias 'c'
```

Trvalé aliasy

Aliasy definované v příkazové řádce jsou dočasné a zmizí po zavření shellu. Pro trvalé aliasy je potřeba je přidat do konfiguračních souborů:

- `.bashrc` v domovském adresáři (pro interaktivní shelly)
- `.bash_profile` (pro přihlašovací shelly)

```
# Přidání do .bashrc
echo "alias ll='ls -la'" >> ~/.bashrc
```

Proměnné

Proměnné jsou způsob, jak ukládat a používat data v shellu.

Pravidla pro pojmenování proměnných

- Může obsahovat písmena, číslice a podtržítko
- Musí začínat písmenem nebo podtržítkem
- Názvy jsou case-sensitive (rozlišují velká a malá písmena)
- Konvence: systémové proměnné jsou VELKÝMI písmeny, uživatelské proměnné malými

Vytváření a přiřazování hodnot proměnným

```
# Základní přiřazení
jmeno="Jan"

# Pozor na mezery - žádné mezery kolem =
jmeno = "Jan"  # Chyba!
```

Přístup k hodnotám proměnných

```
echo $jmeno      # Jednoduché použití
echo ${jmeno}    # Ohraničení jména proměnné (užitečné v některých kontextech)
echo ${jmeno}uv_pes  # Vypíše "Januv_pes"
```

Speciální proměnné shellu

Proměnná	Význam
`$HOME`	Domovský adresář uživatele
`$PATH`	Seznam adresářů, kde shell hledá spustitelné soubory
`$PWD`	Aktuální pracovní adresář
`$USER`	Jméno přihlášeného uživatele
`$SHELL`	Cesta k aktuálnímu shellu
`$RANDOM`	Náhodné číslo mezi 0 a 32767
`$HOSTNAME`	Jméno počítače
`$SECONDS`	Počet sekund od spuštění shellu
`$LINENO`	Aktuální číslo řádku ve skriptu

Typy proměnných v BASH

Skalární proměnné Standardní proměnné, které obsahují jednu hodnotu.

```
jmeno="Jan"
```

Pole (arrays) Proměnné, které mohou obsahovat více hodnot.

```
# Indexované pole (s číselným indexem)
soubory=("f1.txt" "f2.txt" "f3.txt" "f4.txt" "f5.txt")
echo ${soubory[1]}  # Vypíše "f2.txt" (indexuje se od 0)
echo ${soubory[@]}  # Vypíše všechny pruky

# Asociativní pole (hash map) - dostupné od BASH 4
declare -A osoby
osoby[jmeno]="Jan"
osoby[prijmeni]="Novák"
echo ${osoby[jmeno]}  # Vypíše "Jan"
```

Export proměnných

Standardně jsou proměnné "lokální" pro aktuální shell. Pro zpřístupnění proměnné podřízeným procesům (child processes) je nutné ji exportovat.

```
export JMENO="Jan"
```

Exportovaná proměnná je děděna všemi potomky aktuálního shellu.

Příklady práce s proměnnými

```
# Základní operace
x=10
y=20
echo $x $y  # Vypíše: 10 20

# Aritmetické operace
z=$(( x + y ))
echo $z  # Vypíše: 30

# Uložení výstupu příkazu do proměnné
datum=$(date)
echo "Aktuální datum a čas: $datum"
```

Poziční parametry

Poziční parametry jsou speciální proměnné, které obsahují argumenty předané skriptu nebo funkci.

Parametr	Význam
`$0`	Jméno skriptu nebo shellu
`$1` až `$9`	První až devátý argument
`${10}` a vyšší	Desátý a další argumenty (vyžadují složené závorky)
`$#`	Počet předaných argumentů
`$*`	Všechny argumenty jako jeden řetězec
`$@`	Všechny argumenty jako samostatné položky
`$$`	PID (Process ID) aktuálního shellu
`$?`	Návratový kód posledního příkazu (0 znamená úspěch)
`$!`	PID posledního procesu spuštěného na pozadí

Příklad použití pozičních parametrů

```
#!/bin/bash
# Skript, který ukáže použití pozičních parametrů
echo "Jméno skriptu: $0"
echo "První argument: $1"
echo "Druhý argument: $2"
echo "Počet argumentů: $#"
echo "Všechny argumenty: $*"
```

Posouvání pozičních parametrů

Příkaz `shift` posune poziční parametry doleva (odstraní $1, $2 se stane $1 atd.).

```
shift      # Posune parametry o jednu pozici
shift 2    # Posune parametry o dvě pozice
```

Pojem skript, způsoby spouštění skriptu, ladění skriptu

Co je shell skript

Shell skript je textový soubor obsahující posloupnost příkazů, které může shell vykonat. Umožňuje automatizaci opakovaných úloh, spojování příkazů a vytváření vlastních utilit.

Vytvoření shell skriptu

1. Vytvořte textový soubor s příkazy
2. Na první řádek skriptu přidejte tzv. "shebang" - určuje, který interpret má skript zpracovat
3. Nastavte skriptu práva pro spuštění

```
#!/bin/bash
# Jednoduchý skript, který vypíše pozdrav
echo "Ahoj, světe!"
```

Spouštění shell skriptu

1. Explicitní spuštění interpretem

```
bash muj_skript.sh
sh muj_skript.sh
```

2. Spuštění jako spustitelný soubor Po nastavení práv pro spuštění:

```
chmod +x muj_skript.sh
./muj_skript.sh
```

3. Spuštění přes cestu Pokud je skript v adresáři, který je v $PATH:

```
muj_skript.sh
```

4. Spuštění jako skript přímo v aktuálním shellu

```
source muj_skript.sh
. muj_skript.sh  # Zkrácená forma příkazu source
```

Ladění skriptů

Spuštění v debugovacím režimu

```
bash -x muj_skript.sh  # Vypíše každý příkaz před jeho provedením
```

Zapnutí/vypnutí debugovacího režimu v části skriptu

```
#!/bin/bash
echo "Toto není sledováno"
set -x  # Zapnutí debug režimu
echo "Toto je vypsáno s ladícími informacemi"
set +x  # Vypnutí debug režimu
echo "Toto opět není sledováno"
```

Kontrola syntaxe bez spuštění

```
bash -n muj_skript.sh
```

Přidání podrobnějších informací

```
set -v  # Zapíná výpis vstupu shellu
```

Příklad ladění s komentáři

```
#!/bin/bash -xv
# Přepínače -xv zapínají debug režim pro celý skript

# Tento komentář uvidíte ve výstupu díky -v
echo "Test začíná"  # Tento příkaz bude vypsán a pak proveden díky -x

# Kontrola počtu argumentů
if [ $# -ne 1 ]; then
    echo "Chyba: Zadejte přesně jeden argument" >&2
    exit 1
fi

echo "Vše v pořádku"
```

Základní programové konstrukce (podmíněné příkazy, cykly)

Podmíněné příkazy

Konstrukce if-then-else

```
if [ podmínka ]; then
    # příkazy, pokud je podmínka splněna
elif [ jiná_podmínka ]; then
    # příkazy, pokud je jiná_podmínka splněna
else
    # příkazy, pokud žádná podmínka není splněna
fi
```

Poznámka: Mezery kolem hranatých závorek jsou povinné!

Konstrukce test Příkaz `test` nebo jeho alias `[ ]` umožňuje testování podmínek.

Porovnávání čísel

```
[ "$a" -eq "$b" ]  # a je rovno b
[ "$a" -ne "$b" ]  # a není rovno b
[ "$a" -lt "$b" ]  # a je menší než b
[ "$a" -le "$b" ]  # a je menší nebo rovno b
[ "$a" -gt "$b" ]  # a je větší než b
[ "$a" -ge "$b" ]  # a je větší nebo rovno b
```

Porovnávání řetězců

```
[ "$a" = "$b" ]     # a je rovno b
[ "$a" != "$b" ]    # a není rovno b
[ -z "$a" ]         # a je prázdný řetězec
[ -n "$a" ]         # a není prázdný řetězec
```

Testování souborů

```
[ -e "$soubor" ]    # soubor existuje
[ -f "$soubor" ]    # soubor existuje a je to běžný soubor
[ -d "$soubor" ]    # soubor existuje a je to adresář
[ -r "$soubor" ]    # soubor existuje a je čitelný
[ -w "$soubor" ]    # soubor existuje a je zapisovatelný
[ -x "$soubor" ]    # soubor existuje a je spustitelný
[ -s "$soubor" ]    # soubor existuje a není prázdný
```

Logické operátory

```
[ "$a" -eq 1 -a "$b" -eq 2 ]  # AND - obě podmínky musí platit
[ "$a" -eq 1 -o "$b" -eq 2 ]  # OR - alespoň jedna podmínka musí platit
[ ! "$a" -eq 1 ]               # NOT - negace podmínky
```

Zkrácené vyhodnocení podmínky

```
# AND - druhý příkaz se provede, jen pokud první uspěje
[ -d "$dir" ] && echo "Adresář existuje"

# OR - druhý příkaz se provede, jen pokud první selže
[ -d "$dir" ] || mkdir "$dir"
```

Case (switch)

```
case "$promenna" in
    vzor1)
        # příkazy pro vzor1
        ;;
    vzor2|vzor3)
        # příkazy pro vzor2 nebo vzor3
        ;;
    *)
        # příkazy pro cokoliv jiného (default)
        ;;
esac
```

Příklad použití case:

```
#!/bin/bash
# Zjištění typu souboru podle přípony
case "$1" in
    *.jpg|*.jpeg|*.png|*.gif)
        echo "Obrázek"
        ;;
    *.mp3|*.wav|*.ogg)
        echo "Audio soubor"
        ;;
    *.txt|*.doc|*.pdf)
        echo "Textový dokument"
        ;;
    *)
```

```
        echo "Neznámý typ souboru"
        ;;
esac
```

Cykly

Cyklus for

```
# Základní syntaxe
for proměnná in seznam; do
    # příkazy
done

# Příklady seznamů
for i in 1 2 3 4 5; do
    echo "Číslo: $i"
done

for soubor in *.txt; do
    echo "Zpracovávám soubor: $soubor"
done

# C-style for (BASH)
for ((i=0; i<5; i++)); do
    echo "Iterace: $i"
done
```

Cyklus while

```
# Základní syntaxe
while [ podmínka ]; do
    # příkazy
done

# Příklad: Načítání vstupu, dokud není zadáno "konec"
while true; do
    read -p "Zadejte příkaz (konec pro ukončení): " cmd
    if [ "$cmd" = "konec" ]; then
        break
    fi
    echo "Zpracovávám: $cmd"
done
```

Cyklus until

```
# Základní syntaxe - opakuje příkazy, dokud podmínka není splněna
until [ podmínka ]; do
    # příkazy
done

# Příklad: Čekání na vytvoření souboru
until [ -f "/tmp/signal" ]; do
    echo "Čekám na soubor..."
    sleep 1
done
echo "Soubor vytvořen!"
```

Řízení cyklů

```
break     # Ukončí cyklus
continue  # Přeskočí na další iteraci
```

Příklady skriptů

Skript pro dotaz ano/ne

```
#!/bin/bash

OTZK="$1"
while true; do
    printf "$OTZK "
    read ODP
    case $ODP in
        [Nn]|[Nn][Oo]|[Nn][Ee])
            echo "Ne"
            exit 1
            ;;
        [Yy]|[Aa]|[Yy][Ee][Ss]|[Aa][Nn][Oo])
            echo "Ano"
            exit 0
            ;;
        *)
            echo "Neplatný vstup. Zadejte ano nebo ne."
            ;;
    esac
done
```

Skript s ošetřením výstupu

```
#!/bin/bash

# Kontrola počtu parametrů
if [ $# -ne 2 ]; then
    echo "Chyba: Skript vyžaduje přesně dva parametry." >&2
    exit 2
fi

# Kontrola, zda je první parametr neprázdný
if [ -z "$1" ]; then
    echo "Chyba: První parametr nemůže být prázdný." >&2
    exit 1
fi

# Kontrola, zda je první parametr adresář
if [ ! -d "$1" ]; then
    echo "Chyba: '$1' není adresář." >&2
    exit 2
fi

# Zde pokračuje hlavní kód skriptu...
echo "Zpracovávám adresář: $1"
```

Kontrola, zda je vstup číslo

```
#!/bin/bash

# Metoda 1: Pomocí regulárního výrazu
if [[ ! "$1" =~ ^[0-9]+$ ]]; then
```

```
    echo "Chyba: Zadejte celé číslo." >&2
    exit 1
fi

# Metoda 2: Pomocí testu s declare
if ! typeset -i cislo="$1" 2>/dev/null || [ "$cislo" != "$1" ]; then
    echo "Chyba: Zadejte celé číslo." >&2
    exit 1
fi

echo "Zadané číslo: $1"
```

Návratové hodnoty

Návratová hodnota skriptu nebo příkazu je celé číslo, které indikuje úspěch nebo typ chyby. Konvence:

Návratová hodnota	Význam
0	Úspěšné provedení
1	Obecná chyba
2	Nesprávné použití příkazu (např. špatné parametry)
126	Příkaz nelze spustit (např. chybí oprávnění)
127	Příkaz nebyl nalezen
128+	Ukončení signálem (128 + číslo signálu)

Návratovou hodnotu nastavíte pomocí příkazu `exit`:

```
exit 0   # Úspěšné ukončení
exit 1   # Chyba
```

Shrnutí

Shell je mocný nástroj pro práci s unixovými systémy, který umožňuje nejen interaktivní ovládání, ale také programování složitějších postupů a automatizaci úloh. V této kapitole jsme si představili:

1. **Druhy shellů** - zejména rodiny Bourne a C Shell, s důrazem na BASH
2. **Metaznaky** - speciální znaky, které shell interpretuje zvláštním způsobem
3. **Aliasy** - zkratky pro často používané příkazy
4. **Proměnné** - ukládání a manipulace s daty
5. **Poziční parametry** - práce s argumenty skriptů a funkcí
6. **Skripty** - vytváření, spouštění a ladění shellových programů
7. **Programové konstrukce** - podmínky a cykly pro řízení toku programu