Projekt v rámci SIPVZ:

IMPLEMENTACE OPERAČNÍHO SYSTÉMU LINUX DO VÝUKY INFORMAČNÍCH TECHNOLOGIÍ

LINUX

Lekce 15

Shell - 2

Obsah lekce:

Cíle

Po skončení této lekce studenti budou schopni:

- Definovat I/O kanály a I/O přesměrování.
- Vytvořit užitečný alias
- Orientovat se v proměnném prostředí Shellu
- Zvládat operace se Shellem

Shell

- Populární shelly
  - C Shell
  - Ash a Bash
  - Kron Shell
- Historie příkazů
  - Použití kláves UP a DOWN
- Dokončování příkazů
  - Pracuje pro všechny příkazy a adresáře
  - Zmáčkněte TAB pro dokončení/zobrazení možných příkazů

Shell neboli příkazový interpret je program, který překládá příkazy zadané uživatelem do instrukcí pro operační systém. Operační systém Linux je složen za samostatného jádra, všechny ostatní programy, které jsou obsaženy v jednotlivých distribucích, jsou v podstatě jen přídavek pro uživatele.

Dvěma základními charakteristikami shellu je interaktivita a provádění skriptů. Průměrná distribuce obsahuje vždy několik shellů, z nichž dovoluje uživateli vybrat. Každý z nich nabízí jinou variantu syntaxe a interaktivitiy. Ačkoliv mnoho uživatelů používá svůj oblíbený příkazový interpret, Bourne Again Shell (bash) se na většině distribucí používá jako implicitní pro jeho velkou oblíbenost. Bash je pojmenován po tvůrci prvního programovatelného shellu pro Unix jménem Stephan Bourne a používá funkce pro historii příkazů a automatické dokončovaní příkazů. Tento příkazový interpret je používán ve všech dalších příkladech, pokud ovšem není uvedeno jinak.

Populární shelly

První shell pro Unix byl Bourne shell (sh), ačkoliv je to předek všech dalších příkazových interpretů, na Linuxu není běžně dostupný. Přinesl velké změny a nastínil vývoj a funkce dalších interpretů. Od dob svého vzniku se moc nezměnil, ale je stále velmi populární.

C Shell

C-shell byl vyvinut na University of California v Berkeley programátorem jménem William Joy a poskytuje mnoho rozdílných programátorských konstrukcí, přesto existuje možnost kompatibility s Bourne shellem v oblasti interaktivity s uživatelem. Přestože je implementována historie příkazů, nepodporuje editaci příkazů již zapsaných. Autor posléze napsal vylepšený C-shell, ten je znám pod označením TC-shell (tcsh), ten již podporuje editaci příkazů, automatické dokončování, aliasy a jako přídavek též náhradu samotných vnitřních příkazů shellu.

Ash a Bash

Dále se ještě vyvíjejíc další dvě generace Bourne shellu. Jsou to Ash (ash) a Bash (bash). Ash je verze původního Bourne s přidanými vlastnostmi z V shellu. Je to základní shell a většinou se používá při instalacích (poté se nahradí jiným) a na disketových distribucích. Bash byl vyvinut pod GNU projektem (Free Software Foundation) a je to implicitní příkazový interpret pro všechny distribuce. Bash je kompatibilní s Bourne a poskytuje implicitně editaci příkazů a náhrady příkazů, dokončování příkazů, aliasy, vyhledávací utility a také malý textový editor. Na Linuxu odkazuje sh na bash, protože Bourne není dostupný. Bash a T-shell mají také stejné vlastnosti jako historii, kontrolu procesů, nahrazování jmen souborů (linkování).

Korn shell

Korn je shell je jeden z nejpopulárnějších interpretů pro Unix a byl první, kdo importoval moderní techniky C-shellu do Bourne. Poskytuje v základě ty samé funkce jako Bourne a C-shell a ještě mnohem navíc, jeho hlavní výhodou je, že má svého bratříčka, který je ovšem zdarma a dostupný pro Linux a nazývá se Pd-ksh (pdksh).

Z Shell

Jeden z novějších interpretů, které se vyvíjejí je tu Z-shell (zsh), který je kompatibilní s Bourne a poskytuje editaci příkazů a všechny podstatné vlastnosti z Kornu, Bashe a T-shellu. Jestliže se spustí s implicitní konfigurací velmi se podobá Kornu.

Tabulka 9-1 přináší seznam vlastností jednotlivých příkazových interpretů.

	Ash	Bash	C-shell	PD-ksh	T-shell	Zsh
Autor	Kenneth Almquist	Brian Fox, Chet Ramery	William Joy	Eric Gisin	William Joy	Paul Falstand
Binárka	ash	bash	csh	pdksh	tcsh	zsh
Implicitní shell	Ne	Ano	Ne	Ne	Ne	Ne
Kontrola událostí	Ne	Ano	Ano	Ano	Ano	Ano
Aliasy	Ne	Ano	Ano	Ano	Ne	Ano
Funkce shellu	Ano	Ano	Ne	Ano	Ano	Ano
Přesměrování	Ne	Ano	Ano	Ano	Ano	Ano
Historie	Ne	Ano	Ano	Ano	Ano	Ano
Editace příkazů	Ne	Ano	Ne	Ano	Ano	Ano
Dokončování názvů souborů	Ne	Ano	Ano	Ano	Ano	Ano
Dokončování názvů uživatelů	Ne	Ano	Ano	Ano	Ano	Ano
Dokončování historie	Ne	Ano	Ne	Ne	Ano	Ano

Tabulka 15-1 – Různí interpreti příkazů a jejich vlastnosti

Pro identifikaci aktuálního shellu se zadává následující příkaz:

```
$ echo $SHELL
```

Příkaz vytiskne absolutní cestu, kde je aktuální shell alokován.

Pro změnu aktuálního shellu, zadejte jméno shellu, pro který jste se rozhodli, do příkazové řádky. Například pro změnu z Bashe do c-shellu zadáme následující:

```
$ csh
```

Příkazová řádka změní signal dolaru ($) na znak procenta (%), což indikuje, že je aktivní csh shell.

Pokud chceme změnit shell ihned při přihlášení, neboli implicitní shell, zadáme následující:

```
% chsh –s /bin/ash
```

Jakmile restartujeme počítač, použitý shell bude Ash, do té doby než ho opětovně změní uživatel. Abychom mohli pracovat s nějakým shellem, musíme si být jisti, že je daný shell přítomen v systému. Kompletní seznam instalovaných interpretů lze najít otevřením /etc/shells (pokud tento soubor existuje) neb zadáním příkazu:

```
# chsh –l
```

Historie příkazů a automatické dokončování

Nejnovější verze bash, tcsh a zsh nabízejí přehled zadávaných cpříkazů a automatické dokončování příkazů. Historie zaávaných příkazů je dosutpná po zadání klávesových šipek <UP> a <DOWN>, tím se zobrazí předchozí příkaz a těmito klávesami můžeme vždy listovat nahoru nebo dolu

v seznamu již zadaných příkazů. Tento výpis se i může řadit podle počtu zadní daného příkazu, čím častější tím níže bude v seznamu a tím blíže našemu výběru. Zadáním příkazu history lze zobrazit seznam již všech zadaných příkazů a to do té doby než je seznam smazán či se nepřepíše novým obsahem. Změnou proměnné shellu $HISTSIZE lze nastavit počet příkazů, které si bude interpret pamatovat, například:

```
bash$ HISTORY=1000
```

nastaví, aby si bash pamatoval posledních 1000 příkazů.

Automatické dokončovaní příkazů je schopnost dokončovat názvy adresářů, souborů nebo názvů příkazů pomocí klávesy <TAB> tabulátor. Jestliže je zde více možností, shell doplní název tak daleko, jak je unikátní, a dále čeká zase na uživatelovo zadání. Zmáčknutí klávesy <TAB> zobrazí všechny možné názvy, které jsou možné doplnit podle začínajících písmen a uživatel si z nich může vybrat to správné.

Například jestliže adresář obsahuje soubory pojmenovaní julie, angus, meredith, allison a lenny a zadáme bash$ m a zmáčkneme klávesu <TAB> zobrazí se bash$ meredith.

Přesměrování I/O

CLI (Command Line Interface) dovoluje přesměrování vstupu, výstupu a chyb jinam než na standartní vstup (stdin), standartní výstup (stdout) a standardní chyby (stderr) zařízení. Normálně je stdin kanál asociován s klávesnicí a stdout s monitorem počítače

Symboly přesměrování

Je důležité si uvědomit, že ne všechny vstupy pocházejí z klávesnice nebo myši a že je důležité ovládat vstup a výstup programů. Je zde několik speciálních znaků, které mohou být pro tento účel použity. Díky nim přesměrujeme vstup pro stdin nebo výstup pro stdout stejně dobře jako případné chyby, pokud by bylo potřeba. Můžete si představit 0 standardní vstup, 1 pro standardní výstup a 2 pro standardní chyby.

Tabulka 9-2 vypisuje symboly pro přesměrování a jejich funkce.

Znak pro přesměrování	Co to udělá
<0	Přesměruje standardní vstup.
1>	Přesměruje standardní výstup.
2>	Přesměruje standardní chyby.
>	Přesměruje výstup z příkazu na levé straně symbolu do zařízení nebo souboru na pravé straně. Pokud daný soubor existuje, pak ho přepíše. Je to zkratka pro 1>.
>>	Má stejný efekt jako >, ale přidává výstup na konec souboru, takže ho nepřepíše.
<	Zkratka pro <0.
|	Přesměruje výstup na levé straně příkazu do vstupu příkazu na pravé straně. (symbol pipe)

Tabulka 15-2 – Soupis operátorů pro přesměrování

Přesměrování vstupu

Přesměrování dovoluje ovlivnit vstup programu tak, aby mohl přijít z jakéhokoliv zdroje, ne pouze z klávesnice. Například pokud bude existovat program nazvaný printNames, který tiskne seznam jmen a vstup by měl do něj přijít z textového souboru names, pak program vykonáme pomocí takovéto syntaxe:

```
bash$ printNames < names
```

V tomto příkladu obsah souboru names bude odeslán do příkazu printNames jako vstupní data.

Přesměrování výstupu do souboru

Výstupní data příkazu mohou být také odeslána jinam než na obrazovku, jestliže tak chceme. Rozmyslete si funkci následujícího:

```
bash$ ls > /home/directoryListing.txt
```

Pokud zadáme příkaz ls samotný, výstup bude poslán do stdout, který je běžně asociovaný s terminálovou obrazovkou. Avšak v tomto příkladu byl použit operátor přesměrování pro odeslání výstupu příkazu ls do souboru s názvem directoryListing.txt. Rovněž ovšem musíme počítat s jistými omezeními tohoto přesměrování, jestliže nebude tento soubor existovat, pak se vytvoří, což je v pořádku, ale na druhou stranu, pokud již existovat bude a zadáme tento příkaz, jeho obsah se přepíše. Pokud nám toto nevyhovuje, můžeme použít jiného symbolu pro přesměrování a ten nám zajistí, že výsledek bude připojen na konec původního souboru.

```
bash$ ls >> /home/directoryListing.txt
```

Pamatujte si, že takto zapsáno se výpis připojí na konec daného souboru a pokud tento soubor nebude existovat, bude vytvořen.

Přesměrovávání chyb

Někdy se stává, že za běhu nějakého příkazu se vyskytne chyba, její výpis je odeslán do systémových log souborů. Možné vyvstalé chyby během běhu jednotlivých programů mohou být také přesměrovány do libovolného umístění, například do souboru, který za tímto účelem vytvoříme. Prohlédněme si následující:

```
bash$ ls disk{1,2,3} 2> listingError.txt >> /home/directoryListing.txt
```

V tomto příkladu, názvy souborů a adresářů obsažených v adresářích disk1, disk2, disk3 budou zkopírována do souboru pojmenovaného jako direcoryListing.txt, který je umístěn v adresáři /home. Pokud se během tohoto procesu obejví jakákoliv chyba, bude zkopírována do souboru listingError.txt, který je též v adresáři /home.

Piping výstupu

Přesměrování z jednoho programu do druhého je jeden z nejčastějších druhů přesměrování. Pipe (|) (Zajímavé, že? – zkratka <CTRL-ALT-W>) je efekivní pro spjení jednoho programu nebo příkazu na vstup jiného. Pokud je použit tento symbol mezi dvěma příkazy, spojuje výstup prvního příkazu na levé straně na vstup příkazu na pravé straně. Jako zde:

```
bash$ ls –l  | more
```

Toto způsobí, že výstup výpisu adresářů bude vstupem pro příkaz more. Jediným omezením je počet programů, které mohou být takto spojeny a to na základě dostupných systémových zdrojů.

Aliasy a zkratky

- Porozumnění aliasům
  - Krátký název pro dlouhý příkaz
  - Primárně používán pro zmenšení počtu stisknutých kláves
- Klávesové zkratky shellu
  - Editace příkazové řádky
  - Použití operátoru !

Pouze zkušenější uživatelé si dovedou představit jak se jednoduché příkazy mohou stát těžšími k zapamatování, pokud k nim přidáváme velké množství přepínačů. To je přesně ten důvod proč je důležitá implementace aliasů. Klávesové zkratky mohou být další velkou úsporou času uživatele, zvláště pokud pracujete s více soubory a adresáři.

Porozumnění aliasům

Aliasy příkazů poskytují schopnost připojit jednoduše zapamatovatelné jméno komplikovanějšímu a delšímu příkazu.

Schopnost nastavit jistý řetězec tak, aby provedl jinou operaci, je velmi důležitá, pokud pracujete na více platformách. Často uživatelé MS Windows v příkazové řádce Linuxu tápou, protože jsou zvyklí na jisté příkazy a ty zde nefungují. Jednoduchým nastavením aliasu nějakému příkazu si velmi zpříjemníme a usnadníme práci

Šetření počtu stisku kláves

Pro ukážku si představím, že si uživatel přeje vidět celý výpis adresářů pokaždé, co zadá příkaz ls. Použijeme tedy alias, který nám tuto akci připraví:

```
alias ls='ls –l'
```

Poté co je tento alias vvytvořen, pokaždé zadání příkazu ls se provede ve skutečnosti příkaz ls –l.

Pro odstranění aliasu napíšeme:

```
unalias ls
```

Běžné aliasy

Zde je výpis několika běžně používaných aliasů:

```
alias ..='cd ..'
alias …='cd ../..'
alias dir='ls – l'
alias l='ls –alF'
alias la='ls –la'
alias ll='ls – ll'
alias ls='ls $SL_OPTIONS'
alias ls-l='ls –l'
alias md='mkdir –p'
alias o='less'
alias rd='rmdir'
```

Naneštěstí jsou aliasy specifické pro každý uživatelský účet, ale systémový administrátor často přechází mezi stanicemi a uživatelskými účty, zde například nasazuje svoje skripty či provádí úlohy správy systémy, kde nemůže použít svoje vlastní aliasy.

Pokud si nejste jisti, že daný příkaz není alias, který si někdo nastavil, použijte znak backslash (\) před daným příkazem. Takto příkaz \ls nevykoná dané ls $LS_OPTIONS ale pouze jednoduché ls.

Klávesové zkratky

Editace příkazové řádky

Základní vlastností Bashe je schopnost editace záznamu v příkazové řádky. V jakémkoliv bodě může uživatel použít levou a pravou kurzorovou klávesu aby posunul kurzor na jiné místo. Může mazat nebo přidávat znaky nebo používat editační zkratky popsané v další tabulce. Shell uchovává histori zadaných příkazů, ta je dostupná klávesami nahoru a dolu a ty můžeme též posléze editovat.

Příkaz	Popis
<CTRL-T>	Provede výměnu dvou sousedních znaků.
<CTRL-W>	Vyjme a uloží pro kopírování vše co je od kurzotu doleva do nejbližšího konce slova.
<CTRL-U>	Vyjme a uloží pro kopírovaní vše od kurzoru doleva.
<CTRL-Y>	Vloží kopírovaný text.
<CTRL-O>	Požívá se společne s klávesami nahoru a dolu v historii příkazů, provede daný příkaz a zobrazí další.
cd ~	Znak pro domácí adresář – vlnovka. (~)
cd -	Mínus je zástupný znak pro adresář, který byl navštíven před současným.

Tabulka 15-3 – Často používané klávesové zkratky pro shell

Použití operátoru !

Vykřičník (!) je původně invencí C shellu a může být použit ve spojení s dalšími operátory pro volání předchozího příkazu z historie příkazů. Například vykřičník následovaný číslem vyvolá příkaz s daným číslem z historie příkazů. Použití dvojtečky dovoluje vybrat nějaké specifické slovo z příkazu a použít ho společně s dalším příkazem. Slova samotná se číslují zleva doprava, první což je většinou název příkazu má číslo nula. Implementace operátoru ! je zobrazena v tabulce 9-4.

Příkaz	Popis
!!	Zavolá poslední příkaz.
!-5	Zavolá pátý příkáz.
!30	Zavolá třicátý příkaz z historie.
!h	Zavolá poslední příkaz začínající na písemneo h.
!:	Zavolá druhé slovo v posledním příkazu.

Tabulka 15-4 – Využití operátoru !

Proměnné prostředí/Shellu

Prostředí shellu je definováno nastavením několika parametrů, které ovládají vzhled, prostředí, interaktivitu shellu a všechny mohou být přizpůsobeny podle uživatelovo přání. Proměnné prostředí jsou podobné k proměnným shellu, u obou se jedná o řetězec, proměnná prostředí prochází do všech podřízených procesů při jejich inicializaci. Rozdíl v nich je ve viditelnosti těchto nastavení v a mimo program shellu.

Proměnné prostředí jsou globální proměnné, a každý shell k nim přistupuje. Proměnné shellu jsou lokání proměnné, jejich viditelnost je omezena na shell, ve kterém jsou umístěny.

Konvence pojmenovávání

Pro pojmenovávání proměnných se používají velká písmena pro proměnné prostředí a malá i velká písmena pro proměnné shellu, ty navíc mohou obsahovat znaky, číslice, ale nemohou začínat číslem nebo nepísmenným znakem. Velká písmena jsou také používána pro specifikování vnitřních proměnných a exportovaných proměnných. Uživatelé mohou přiřadit nějakou hodnotu své proměnné zadáním:

```
bash$ varName=abcdef
```

Export promenných

Proměnná shellu, která je pevně stanovená a je používána v jednom shellu nemůže být použita v jiném, leda že by byla exportována pomocí příkazu export, který má tuto syntaxi:

```
bash$ export varName
```

Proměnná varName bude exportována a zobrazena a může být přiřazena jako hodnota jiné proměnné v jiném shellu. Proměnné shellu mohou být také nastaveny pouze pro čtení, takže ty za normálních okolností nezměníme.

Prohlížení obsahu proměnných pomocí echo

Pokud provádíme přiřazování proměnné, jako například v příkladu nahoře, značka dolaru ($) nepředchází proměnnou. Avšak ve všech ostatních případech a použitích proměnné musíme její název uvodit dolarem. Příkaz echo nám zobrazí aktuální hodnotu dané proměnné:

```
bash$ echo $varName
```

shopt

Verze bashe 2.0 a vyšší přichází s velkým počtem zabudovaných příkazů. Jedním z těchto built-in příkazů je shopt, který je konfiguračním nástrojem pro shell. Zadáním shopt –s option nebo shopt –u option, nastaví nebo naopak zruší nastavení proměnných. Dvě z options jsou cdspell a nocasglob. Pokud je zadána volba cdspell, bash automaticky automaticky opravuje chyby v zápisu písmen v názvech adresářů, které uživatel zadává například po příkazu cd. Volba nocasglob způsobí plné přepnutí bashe do módu, kdy nepočítá s rozdílnými velikostmi písmen. kompletní seznam voleb a možností získáte po zadání příkazu shopt.

Otázky k opakování

Lab