Projekt v rámci SIPVZ:

IMPLEMENTACE OPERAČNÍHO SYSTÉMU LINUX DO VÝUKY INFORMAČNÍCH TECHNOLOGIÍ

LINUX

Lekce 16

Shell - 3

Obsah lekce:

Cíle

Po skončení této lekce studenti budou schopni:

- Vyjmenovat koncepty multi-taskingu pro shell (shelly a subshelly).
- Orientovat se v multi-taskingových procesech
- Přizpůsobit si příkazovou řádku
- Napsat a provést jednoduchý skript pro shell.

Přizpůsobení Bash

- Proměnná $PATH
  - Kde systém hledá spustitelné
- Velká diskuze ohledně tečky
  - Aktuální adresář často není v cestě
  - Pomáhá zneškodnit Trojské koně
  - Vyřešeno pomocí **./program_name**
- Rozšiřující názvy souborů

Jak již bylo řečeno, je bash velmi přizpůsobitelný a modifikovatelný. Tato sekce popisuje základní možnosti nastavení jednotlivých proměnných aby si člověk mohl ušít svůj bash přímo na tělo.

Proměnná $PATH

Proměnná PATH je obsahuje názvy cest oddělených dvojtečkami, v těchto cestách hledá bash spustitelné soubory, když je shell požádán o jejich spuštění. Pro zobrazení nastavení této proměnní stačí zadat správně příkaz echo:

```
bash$ echo PATH
```

Použitím tohoto příkazu dostanete něco podobné tomuto:

```
/usr/local/bin:/usr/bin:/bin:/usr/home/meredith/bin
```

Jestliže je uživatel ve svém domovském adresáři a chce spustit program v /usr/bin, může to provést několika způsoby. Jednou možností je přímo zavolat program s plnou cestou k němu:

```
/usr/bin/myProgram
```

Další možností je zadat pouze myProgram do příkazové řádky a bash prohledá všechny adresáře specifikovaní v proměnné $PATH jestli se tam nenachází právě náš daný program. Plná cesta k souboru ovšem musí být zadána, pokud se program nenachází v aktuálním adresáři nebo pokud není specifikována v proměnné $PATH. Můžeme též zadat ./myProgram, ale toto je zkratka.

Kromě proměnné $path je zde ještě několik důležitých proměnných, které se velmi často používají. Tabulka 9-5 zobrazuje některé z nejdůležitějších proměnných.

Proměnná	**Popis**
SUSER	Uživatelské jméno aktuálního uživatele.
SUID	UID aktuálního uživatele.
SSHELL	Aktuální shell.
SHOME	Domácí adresář aktuálního uživatele.
SDISPLAY	Umístění zobrazovacího zařízení.
SHOSTNAME	Název počítače.

Tabulka 16-1 – Různé proměnné prostředí.

Velká diskuze ohledně tečky

Logické odůvodnění diskuze ohledně této sekvence (./) přichází díky možným bezpečnostním rizikům. Ve výsledku většina distribucí nezadává do proměnné $PATH znak tečky (.). Bezpečnostní problém se vyskytuje v případě provádění rutin zabudovaných v systémových adresářích /bin/ a /usr/bin/, například příkaz ls. Jestliže je tečka zadána v proměnné $PATH před systémovým adresářem a v adresáři, ve kterém se právě nalézáme, je soubor se stejným názvem jako utilita systému, je spuštěn tento soubor a ne daná utilita, to poskytuje jisté možnosti umístění trojského koně do systému. Například hacker umístí do adresáře svůj program s názvem ls. Pokud uživatel bude chtít vypsat obsah adresáře zadá příkaz ls, ale ve skutečnosti se spustí trojský kůň, který například zobrazí zprávu o chybě a požádá uživatele o nové přihlášení a tím získá jeho heslo:

```
Session error –terminating program
Login:
Password
```

Jako odpověď na tuto zprávu, naivní uživatel zadá svoje jméno a heslo. To dovolí trojskému koni získat jednoduše uživatelovo heslo a to může potom někam odeslat či uložit pro pozdější přečtení hackerem. Poté je každé další zadání příkazu ls provedeno podle normálu a uživatel nepozná, že se něco stalo. Z tohoto důvodu není tečka obsažena v proměnné $PATH. Některé distribuce ji do této proměnné zadávají, ale ne u systémových adresářů.

Rozšiřující názvy souborů

Nejvíce používaný způsob pro identifikaci souboru je napsat jeho plné jméno. Avšak jsou tu i jiné možnosti jak určit soubor pomocí meta charakterů a rozšiřujících názvů. Příkladem metacharakterů jsou otazník (?), hvězdička (*) a závorky ([]). Metacharaktery nebo zástupné masky (wildcards). Pokud jsou tyto znaky přidány do shellu, nařizují mu zahrnout do výběru všechny soubory, které vyhovují zadané podmínce. Vypadá to zhruba následujíc:

```
$ls
FILE   FILE2  FILE4  file     file2    file40
FILE1  FILE3  FILE5  file1    file3
```

Příkaz ls vypasal všechny soubory v adresáři kromě skrytých.

```
$ls file
file
```

Příkaz ls vyhledá soubor s název file.

```
$ls file*
file      file1     file2     file3     file40
```

Příkaz ls rozeznal zástupnou masku (*) a vyhledal všechny soubory začínající na frázi file.

```
$ls file?
file1     file2     file3
```

Zástupná znak (?) representuje jeden znak a tudíž příkaz ls vyhledal soubory začínající na frázi file a po něm jedno další písmeno.

```
$ls file[0-9] file[0-9][0-9]
file1     file2     file3     file40
```

První závorka obsahuje rozmezí znaků 0,1,2,3,4,5,6,7,8,9 a to v kombinaci s řetězcem file. Druhá a třetí závorka nastavují stejnou masku, ale hned dvakrát za sebou takže se jedná o interval od 00 do 99.

Každý zástupný znak způsobuje jinou akci a mohou být v podstatě libovolně kombinovány.

```
$ls ????[0-9]
FILE1  FILE2  FILE3  FILE5  file1     file2     file3
```

Skriptování Shellu

Krom běhu programů, příkazy shellu dovolují použití různých proměnných, testování proměnných a větvení programu podle jejich hodnot. Formát proměnných a jejich testování závisí na použitém shellu. Jak již bylo řečeno sh, bash, ksh, zsh zaměstnávají množství kontrolních konstrukcí (například for a while smyčky).

Tvorba skriptů pro shell:

Mnoho shellů používá programovacích konstrukcí, jako jsou smyčky a podmínky. Avšak pro provedení jednoduchých instrukcí nám stačí pouze seznam příkazů.

Prvně si vytvoříme spustitelný soubor:

```
$ >sam
$ chmod +x sam
```

Otevřeme ho v textovém editoru a zadáme následující:

```
#!/bin/bash
# Toto je příklad skriptu
        cd /etc
        ls – la
        ps aux
        echo "Toto je vzorovy skript."
```

Křížek je symbol pro označení komentáře, a vše co je za ním se ignoruje ovšem s jedinou výjimkou. Ta je všeobecně známa jako hash-bang hack (#!), což je konstruktorem interpretem jak prefix pro označení spustitelného skriptu a nařizuje shellu provést následující řádky. Skript poté vykoná všechny příkazy, které obsahuje po jednotlivých řádcích.

Spuštění skriptů

Jakmile se jednou vytvoří skript, jsou zde potom čtyři možnosti jak ho spustit. První je vytvořit ze skriptu spustitelný soubor, použitím příkazu chmod a poté spuštěním z rodičovského adresáře zadáním ./filename. Pro předchozí příklad byly již práva nastaveny při vytváření souboru, pro jeho spuštění tedy stačí zadat ./sam.

Druhý způsob je vytvoření spustitelného souboru. Absolutní cesta souboru může být zkopírovana do proměnné $PATH, což dovolí provádět skript z jakéhokoliv adresáře. Nejjednodušší cestou je vytvořit podadresář /bin v uživatelově domácím adresáři a změnit $PATH následujícím příkazem.

```
$ PATH=$PATH:/mnt/home/luke/bin
```

Všechny další skripty mohou být přesunuty do /mnt/home/luke/bin adresáře a poté mohou být vyvolány z jakéhokoliv adresáře. Další dvě zbývající možnosti jsou použití příkazu source a exec.

Zdroj

Příkaz source je zabudovaný příkaz shellu bash, který přečte a spustí příkazy ze souboru. Pokud je zadán příkaz source, prohledá se PATH adresář pro daný soubor. Pokud jsou s tímto příkazem zadány nějaké další parametry, předpokládá se, že se jedná o parametry pro jednotlivé příkazy uvnitř skriptu.

Exe (binárka)

Zde se také jedná o vnitřní zabudovaný příkaz samotného shellu, který může spustit daný příkaz bez vytvoření dalšího procesu a může přesměrovat standardní vstup, výstup i chyby samotného skriptu z něj i do něj. Pokud ovšem spouštíme příkaz, který není zabudován v shellu, vytvoří se nový proces v závislosti na nastavení rodičů procesu. Bohužel spuštění příkazu exec sebere současnému procesu prioritu. Souhrn použití zástupných znaků u příkazu exec je zobrazeno v další tabulce.

Atributy	(.)	Exec
Může spouštět skripty a kompilovat programy:	Pouze skriptuje	Skriptuje a kompiluje
Vrátí kontrolu do původního procesu:	Ano	Ne
Dává přístup k lokálním proměnným:	Ano	Ne

Tabulka 16-2 – Kontrast (.) a příkazu exec

Úprava souborů .bash

Jednim z prostředků, které nám ulehčí práci v shellu je možnsot jeho nastavení. Prostředí shellu může být překonfigurováno pomocí některého z konfiguračních souborů .bash_profile, .bash_login nebo .profile. Soubor .bash_profile je konfigurační soubor s implicitní konfigurací pro bash, .bash_login je převzat z konfiguračního souboru login a .profile byl převzat z ostatních shellů. Systém se pokouší nalézt tyto souboru v pořadí v jakém byly jmenovány.

Do těchto souborů lze přidávat jednotlivé odkazy na skripty, které se provedou při přihlášení či odhlášení, upravovat lze barvu jednotlivých textů v shellu, provádění příkazů, podrobnější informace naleznete v dokumentačním projektu.

Přizpůsobování vašich příkazových řádek

- Proč měnit?
  - Informace, bezpečnost, návyky
- Proměnné PS
  - PS1 je normální, každodenní shell
  - PS2 je pro dlouhé zalamované příkazy
- Úprava .bashrc pro obsažení:
  - PS1="\u@\h \W\\#" toto zobrazí
  - **username@hostname /path#**

Proč měnit?

Někdy z důvodu lepší komfortnosti používaní, jindy protože si chcete zapamatovat za jakého uživatele jste přihlášení. Někteří administrátoři ještě chtějí vidět aktuální čas, nebo cestu, kde se nacházejí a podobně.

Proměnné PS

Proměnné označené jako PS1 a PS2 definují informace zobrazené uživatele i v jeho primárním a sekundárním promptu po přihlášení. Příkazová řádka zobrazí po přihlášení většinou primární část.

Tyto proměnné jsou většinou uchovány v /etc/bashrc konfiguračním souboru, který obsahuje implicitní proměnné. Jestliže není #PS definována v uživatelovo domácím adresáři /home/usrename/.bashrc potom shell nastartuje s implicitními hodnotami, ovšem pokud jsou zde definovány, pak použije definované uživatelem, pouze v případě, že by bylo nekorektní, pak se vrátí zase k implicitním.

Proměnná PS akceptuje téměř všechny znaky, které jsou na klávesnici zadat a také několik speciálních kombinací znaků, které se používají pro zvláštní účely, jak si ukážeme později.

Částečný výpis speciálních znaků, které můžete použít v proměnných PS je uveden v tabulce 9-7. Další popisky a další speciální znaky naleznete v dokumentaci.

Speciální znak	**Zobrazí v příkazové řádce**
\d	Zobrazí datum.
\H	Název počítače a domény.
\h	Název počítače.
\n	Číslo řádky.
\s	Jméno shellu.
\t	Aktuální čas v 12 hodinovém HH:MM:SS formátu.
\@	Aktuální čas v am/pm.
\u	Jméno uživatele.
\w	Aktuální pracovní adresář.
\W	Cestu k aktuálnímu adresáři.
\#	Číslo současného příkazu.
\!	Číslo příkazu podle histire příkazů.
$	Jestliže je UID uživatele 0 pak zobrazí #, jestli ne $.
\\	Vytiskne zpětné lomítko.
\[	Začátek netisknutelnch znaků, jako řídící příkazy.
\]	Konec netisknutelných znaků.

Tabulka 16-3 – Speciální znaky v proměnných PS

Příklady příkazových řádek

Předpokládejme dvě deklarované definice PS v uživatelově /home/username/.bashrc souboru:

```
PS1="\u@\h \W\\#"
PS2="\h\\>"
```

Implicitní příkazová řádka definována v prvním řádku bude, obsahovat uživatelovo jméno, následováno znakem zavináče, názvem stanice, současným pracovním adresářem a křížkem (#). Příklad následuje:

```
jaravisen@matrix /etc#
```

Druhý systémový prompt, který byl definován na druhé řádce, zobrazí pouze název stanice a šipku, jako zde:

```
matrix>
```

Druhý uvedený příklad se používá, pokud je příkaz moc dlouhý a rozdělí se na více řádek. Na první bude zobrazena standardní výzva a na druhé tato zkrácená. Pokud chcete pokračovat na další řádku při psaní příkazu, zadejte do příkazu znak lomítka (\) což symbolizuje vstup na nový řádek, pro příklad se to provede takto:

- Zadáme ls |\
- Zmáčkneme enter
    - Poznámka: Nyní se zobrazí PS2.
- Zadáme less
- Zmáčkneme enter
    - Poznámka: Pro konec zadáme q

Jak vidíte, provedl se příkaz ls | less, vypal obsah aktuálního adresáře a výstup poslal do stránkovače jménem less. Příkaz byl zapsán na dvou řádkách, ale jelikož byl zadán znak lomítka \ na konec první řádky, shell správně spojil obě řádky dohromady. Tak se vykonal právě jeden příkaz.

Kontrola multi-taskingových procesů

- Procesy na pozadí a na popředí
    - Je možné mít pouze jeden proces na popředí
    - "program &" - pošle proces na pozadí
- Návrat do příkazové řádky
    - Použití CTRL-Z pro uspání, bg [enter] pro přenesení do pozadí
- Kontrola běžících úloh
    - Zadání jobs pro zobrazení
    - Zadání fg# daný program do popředí

Jednou z nejmocnějších vlastností Linuxu je schopnost dělat více akcí najednou, tento proces se nazývá multi-tasking. Existuje více způsobů jak může uživatel využít multitasking, ty nejhlavnější jsou pospány dále.

Procesy na pozadí a na popředí

Pokud pracujete v příkazové řádce, jako uživatel často nemáte čas čekat až nějaký déle trvající program dokončí svoji práci, před tím než zase budete moc zadávat příkazy. Programy mohou být spouštěny na pozadí zadáním názvu programu a ampersandu (&). Aktuální program běží na popředí v shellu, zatímco program na pozadí běží v sub shellu. Lze také využít přepínaní do jiných konzolí.

Například následující příkaz spustí na pozadí hledání v celé adresářové struktuře hledání souboru obsahující řetezec flename:

```
bash$ find / -name filename &
```

Jestliže si otevřete shell a zadáte locate log > /logs.txt, terminál nebude chvilku reagovat, protože mu bude trvat zpracování příkazu. To je typické pro všechny náročnější procesy na popředí. Jestliže ten samý příkaz locate zadáme s ampersandem (&), příkazová řádka nebude otálet a přenese provádění příkazu na pozadí, zatímco my můžeme okamžitě pracovat dále. Proces se zpracuje na pozadí a my na něj nemusíme čekat.

Zatímco také na popředí může běžet pouze jediný příkaz, na pozadí jich může běžet v podstatě neomezené množství v závislosti na nastavení a výkonu počítače.

Návrat do příkazové řádky

Po spuštění příkazu, pokud trvá déle jeho provádění, než uživatel původně čekal, příkaz může uspat zadáním <CTRL-Z> a odeslat ho na pozadí příkazem bg. Ukažme si to na jednoduchém příkladu:

```
bash$ find / -name filename –log
<CTRL>Z
bash$ bg
bash$
```

Jakmile je daný příkaz na pozadí ukončen, zobrazí se o tom na obrazovce zpráva.

Kontrola běžících úloh

Pro kontrolu běžících procesů nebo úloh či pro přehled počtu běžících programů, stačí zadat následující příkaz:

```
bash$ jobs
```

Jednotlivé úlohy jsou zobrazeny v očíslovaném pořadí. Výpis je organizován podle toho jaký příkaz byl vykonán jako první a tudíž poslední položka na seznamu je příkaz vykonaný jako poslední. Po dokončení jednotlivých úloh se čísla podle toho mění.

Jakmile přebírá proces na pozadí kontrolu nad terminálem, současný proces musí být převeden na pozadí. To můžeme provést příkazem fg. Předpokládejme, že výpis aktuálních procesů bude vypadat zhruba takto:

```
[1]    Running      locate \* > /dev/null &
[2]-   Running      locate log > /dev/null &
[3]+   Running      locate file > /dev/null &
```

Poznámka: Tyto příkazy mohou být spuštěny na jakémkoliv počítač s OS Linux.

Pro přenesení procesu číslo 2 do popředí se použije následující příkaz:

```
bash$ fg %2
```

Otázky k opakování

Lab