Unixové systémy a shell

zápisky z přednášek a cvičení

Jakub Karel (jakub.karel03@upol.cz)
Josef Beťák (josef.betak01@upol.cz)

23. března 2015

Abstrakt

Tento dokument vzniká jako přepis z mých zápisků z přednášek a cvičení předmětu Unixové systémy a shell vyučovaném katedře informatiky na Přírodovědecké fakultě Univerzity Palackého v Olomouci panem Mgr. Janem Outratou, Ph.D. Použití pouze na vlastní nebezpečí – výskyt chybných informací není vyloučen.
Děkuji kolegovi Josefu Beťákovi za jeho usilovnou práci při psaní důkladných poznámek a námětů k tomuto dokumentu.

Obsah

1 První intermezzo

1.1 Použité prostředí

K vytvoření tohoto dokumentu byl použit úžasný typografický nástroj LaTeX 2_ε. Jako testovací operační systém jsem použil Debian 7.7.0 virtualizovaný v programu VMware Workstation nad operačním systémem Windows 8.1.

1.2 Použité konvence

Všechny texty, příkazy, parametry, skripty... Zkrátka vše, co se zadává do terminálu a zároveň všechny odpovědi terminálu, budou vysázeny následujícím způsobem.

```
ping google.com
```

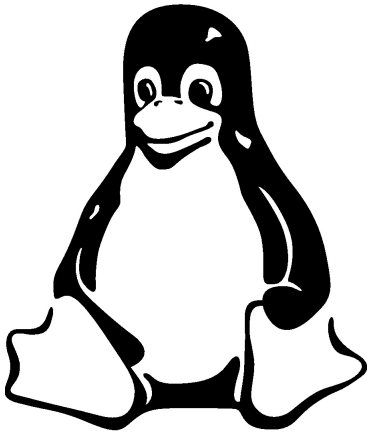
Přestože každý nový řádek v shellu vypadá nějak takto...

```
justify@debian:~$
```

... budu v dalších ukázkách tuto část, ač je velice důležitá, pro zjednodušení vynechávat. To, že jsou všechny příkazy zakončený stiskem klávesy enter, asi není nutné zdůrazňovat. Parametry představené u některých funkcí rozhodně nejsou všechny parametry, které tyto funkce umí zpracovat. Dokonce i funkce, které jsou představeny jako funkce bez parametrů, jich umí většinou řadu zpracovat. Úroveň hloubky zkoumání tedy odpovídá aktuálním výukovým potřebám. Další informace je možné získat z manuálových stránek.

1.3 Errata

Pokud v dokumentu narazíte na chybu, prosím kontaktujte jednoho z autorů. Vaše námitky prověříme a chyby opravíme. Protože dokument vzniká iterativním způsobem, je možné, že dojde k duplicitnímu výskytu některých informací.



Obrázek 1: Tux – maskot systému

2 Základy systému UNIX a GNU/Linux

2.1 Úvod

Jedním z nejvýznamnějších zástupců unixových operačních systémů je systém GNU/Linux. Tento systém je velice populární a hlavně rozšířený. Můžeme ho najít například v clusterech, serverech, PC, tabletech, v mobilech a v dalších zařízeních. I přes velkou popularitu ale zůstává systém nepochopen a to především z hlediska základních principů, filosofie a uživatelských návyků. Většina „řadových“ uživatelů se tak při svém používání setká pouze se zlomkem možností, které se ukrývají pod grafickým uživatelským rozhraním.

Unix je prvním operačním systémem, který byl napsán ve vyšším programovacím jazyce – v jazyce C (1973). Do té doby se programy napsané ve vyšších programovacích jazycích považovaly za příliš pomalé na to, aby mohly sloužit jako operační systém. Výměna assembleru za jazyk C dovolila programovat větší a propracovanější projekty, které se lépe udržovaly a opravovaly. Přestože je koncepce systému Unix stará přes 40 let, stále je funkční a inspirující.

Při práci se systémem si všimnete jedné zajímavosti. Unixové systém jsou „tiché“. To znamená, že při úspěšném vykonání nějaké operace ve většině případů uživateli nesdělí, že operace proběhla v pořádku. V případě, kdy došlo k chybě tuto chybu pochopitelně zobrazí. Toto chování vychází z historických potřeba a možností tehdejších zobrazovacích zařízení a uživatelů. Když se ještě výstup z počítače tisk na tiskárnu, bylo pochopitelně žádoucí, aby množství vytištěného textu bylo co nejmenší.

2.2 Ovládání terminálu

Pro pohyb v řádku v okně terminálu slouží šipky ← a →. Klávesy delete a backspace slouží k mazání znaků na pozici kurzoru, respektive na pozici před kurzorem. Klávesa home slouží pro přechod na začátek řádku a klávesa end na jeho konec. Šipky ↑ a ↓ slouží k pohybu v historii příkazů

2.3 Klávesové zkratky

Klávesa C v následující tabulce reprezentuje klávesu CTRL.

C + L	Smaže obrazovku shellu (lze také použít příkaz clear).
C + K	Text od pozice kurzoru do konce řádku přesune do schránky a smaže.
C + Y	Vloží do řádku text, který byl uložen do schránky příkazem CTRL + K.
C + R	Spustí interaktivní prohledávání historie příkazů.
C + A	Přejde na začátek řádku.
C + E	Přejte na konec řádku.
C + D	Smaže znak na pozici kurzoru.
C + G	Provede ukončení akce (neplést s ukončením probíhající akce).
C + U	Smaže znaky od začátku řádku do pozice kurzoru.
C + T	Zamění poslední dva znaky a posune se vpravo.
C + _	Krok zpět – undo.
C + ALT + T	Spustí terminálové okno.
C + SHIFT + Q	Uzavře terminálové okno.

Tabulka 1: Užitečné klávesové zkratky

2.4 Základní příkazy v shellu

Zadávání příkazů je „case sensitive“ – to znamená, že striktně záleží na velikosti zadávaných znaků. Příkaz „ping“ bude fungovat, zatímco příkaz „Ping“ už ne. Většině příkazů lze při volán předat řadu vstupních argumentů. Ty začínají buďto jednou pomlčkou, která je následována obecně několika málo znaky nebo dvěma pomlčkami, kde jsou to celá nezkrácená slova.

V případě nouze se jako velice zručná klávesa ukazuje tabulátor, protože dokáže doplnit rozepsané jméno programu, souboru nebo adresáře. Při první stisku doplní jméno v případě, že je na výběr pouze jedna možnost. Pokud nedoplní, pak je možností víc. Další stisk klávesy tab pak vypíše všechny možnosti, kterými můžeme pokračovat.

Doplňování cesty klávesou tab je velmi mocné. [Vych03]

Z počátku je nutné seznámit se s poměrně velkým množstvím příkazů a programů, ale v začátcích je nutné se naučit alespoň tyto základní, jinak bude naše práce zdlouhavá a otravná, protože se budeme muset neustále dívat do materiálů, prohledávat manuálové stránky nebo hledat na internetu.

2.4.1 logout

Pokud se nejedná o grafickou emulaci terminálu ale o plnohodnotný shell, provede odhlášení ze systému. V opačném případě uživateli zahlásí:

```
logout
bash: logout: not login shell: use ‘exit’
```

2.4.2 exit

Ukončí grafickou emulaci terminálu (okno terminálu v GUI). Při zavolání vrací své jméno, které se do okna terminálu vypíše ještě předtím, než se okno ukončí. Obecně funkce exit ukončuje předaný proces.

```
exit
exit
```

2.4.3 passwd

Slouží ke změně hesla aktuálně přihlášeného uživatele. Nejprve jste požádání o současné heslo, poté o nové a poté ještě o zopakování nového heslo pro potvrzení.

```
passwd
Changing password for justify.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Můžete si všimnout, že UNIX vám pro jistotu připomene, pro jakého uživatele to vlastně měníte heslo. Při zadávání nejsou vypisovány žádné znaky (toto chování lze přenastavit tak, aby systém vypisoval zadaný zástupný znak), takže se nelekejte, že kurzor zůstává na místě.

2.4.4 echo

Jak už název trochu napovídá, tento program vezme cokoliv, co je mu předáno jako vstupní argument, a vypíše to jako svůj výstup do terminálu.

```
echo lorem ipsum
lorem ipsum
```

2.4.5 who

Funkce who vypíše informace o aktuálně přihlášených uživatelích. a jejich spuštěných terminálech.

```
who
justify  tty8         Feb 23 08:58 (:0)
justify  pts/0        Feb 23 10:27 (:0)
```

2.4.6 w

Pracuje podobně jako who, ale zobrazuje i co uživatelé dělají – jejich procesy a procesorový čas.

```
w
10:58:11 up  4:27,  2 users,  load average: 2.28, 1.84, 1.76
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
justify  tty8     :0               08:58    6days  6.45s  0.12s gdm-session-wor
justify  pts/0    :0               10:27    3.00s  0.13s  0.02s w
```

2.4.7 whoami

„Who am I?“ tedy „Kdo jsem?“ – odpovědí systému je jméno uživatele, které se ptá. Pokud marně přemýšlíte nad tím, kde se takový příkaz dá využít, představte si správce, který má přihlášených několik terminálů, každý pod jiným uživatelem, a potřebuje na nich provádět změny. Takto snadno zjistí, v čím terminálu se právě nachází.

```
whoami
justify
```

2.4.8 groups

Vypíše skupiny, do kterých je uživatel přiřazený.

```
groups
justify cdrom floppy audio dip video plugdev scanner bluetooth netdev
```

2.4.9 uptime

Zobrazí informace o systému, aktuální čas, jak dlouho systém běží, počet přihlášených uživatelů a jeho zatížení v poslední 1, 5 a 15 minutách.

```
uptime
11:11:31 up  4:40,  2 users,  load average: 1.71, 1.88, 1.77
```

2.4.10 date

Provede jednoduchý výpis aktuálního datumu a času.

```
date
Mon Feb 23 11:12:40 CET 2015
```

2.4.11 man

Příkaz man je velice důležitý! Slouží k zobrazení manuálových stránek jednotlivých funkcí systému, kde lze najít veškeré informace o vstupních parametrech, popis, návratové hodnoty a často dokonce i hlubší náhled do funkcionality samotné funkce. I příkaz man má svou manuálovou stránku.

```
man man
```

nebo třeba

```
man ping
```

Pro ukončení zobrazené manuálové stránky se používá klávesa q. Manuálové stránky jsou obvykle rozdělené na 8 očíslovaných sekcí s tím, že parametrem můžeme specifikovat, ve které sekci se mají stránky hledat (následující tabulka platí pro BSD Unix a Linux).

1. obecné/uživatelské příkazy
2. systémová volání
3. funkce knihovny jazyka C
4. speciální soubory (obvykle zařízení nacházející se v /dev) a ovladače
5. formáty konfiguračních souborů a obecné zásady
6. hry a spořiče obrazovky
7. různé
8. příkazy systémové administrace a deamons

```
man 8 ping
```

2.4.12 whatis

Prohledává manuálové stránky a vrací krátký popis – text NAME z manuálové stránky.

```
whatis ping
ping (8)              - send ICMP ECHO_REQUEST to network hosts
```

2.4.13 apropos

Apropos je opět funkce na prohledávání manuálových stránek a jejich DESCRIPTION částí. Pokud najde ve stránce text předaný v parametru, vrací jméno celé stránky na výstup.

```
apropos ping
Compose (5)           - X client mappings for multi-key input sequences
blkmapd (8)           - pNFS block layout mapping daemon
getkeycodes (8)       - print kernel scancode-to-keycode mapping table
iagno (6)             - A disk flipping game derived from Reversi.
l2ping (8)            - Send L2CAP echo request and receive answer
```

```
loadunimap (8)       - load the kernel unicode-to-font mapping table
mapscrn (8)          - load screen output mapping table
minissdpd (1)        - daemon keeping track of UPnP devices up
nping (1)            - Network packet generation tool / ping utility
ntfs-3g.usermap (8)  - NTFS Building a User Mapping File
ping (8)             - send ICMP ECHO_REQUEST to network hosts
...
```

2.4.14 help

Vypíše krátkou nápovědu k shellu se základním příkazy a jejich syntaxí.

2.4.15 type

Vrací, zda je předaný parametr příkaz nebo program. Pokud je to příkaz, pak je zahashovaný. Příkaz type nemá manuálové stránky.

```
type ping
ping is hashed (/bin/ping)
```

Ping je tedy příkaz – dokonce vidíme, kde se nachází na disku. Oproti tomu například editor Bluefish...

```
type Bluefish
bash: type: Bluefish: not found
```

už příkaz není. Je to program.

2.4.16 info

Prohledávání a prohlížení hypertextové dokumentace.

2.4.17 pwd (print working directory)

Vytiskne jméno aktuálního/pracovního adresáře.

```
pwd
/home/justify
```

2.4.18 cd (change directory)

Změna aktuálního adresáře na adresář zadaný cestou – může být jak relativní, tak absolutní. Kromě ukázkových příkladů najdete další užitečné tvary příkazu cd v tabulce 2.

```
justify@debian:/$ cd /home/justify/Dokumenty/
justify@debian:~/Dokumenty$
```

pro přechod do adresáře daného absolutní cestou

```
justify@debian:~/Dokumenty$ cd OS2/
justify@debian:~/Dokumenty/OS2$
```

pro přechod do adresáře daného relativní cestou

Příkaz	Cíl
cd ..	rodičovský adresář
cd /	kořenový adresář
cd	domovský adresář
cd -	předchozí adresář

Tabulka 2: Další použití příkazu cd

Příkaz	Cíl
-	obyčejný soubor
d	adresář
l	odkaz
c	speciální odkaz
s	socket
p	pojmenovaná roura
d	blokové zařízení

Tabulka 3: Typy souborů v Unixových systémech

2.4.19 ls

Příkaz ls bez parametrů vypíše pouze obsah aktuálního adresáře. Přidáním parametrů můžeme docílit vypsání dalších informací o souborech, třeba informace o nastavených přístupových právech, vlastníkovi a časové známce – čas poslední modifikace. Program ls standardně nevypisuje neviditelné soubory (což jsou soubory začínající tečkou), o jejich vypsání musíme požádat přidáním parametru -a.

Přepínačem -F dosáhneme dalšího zpřesnění výpisu – za jméno se doplní jeden další znak, například lomítko / pro adresář nebo hvězdičku * pro spustitelný soubor. Jak už se pomalu stává zvykem, přepínačem -R dosáhneme rekurzivního výpisu obsahu adresářů.

```
ls -l
total 12
drwxr-xr-x 4 justify justify 4096 Feb 18 00:49 OS2
drwxr-xr-x 5 justify justify 4096 Dec  6 19:42 PS
drw-r-xr-x 2 justify justify 4096 Feb 23 16:02 pokusy
```

Zde se trochu víc zaměříme na první písmeno na každém řádku výpisu. První písmeno nám specifikuje, o jaký typ souboru se jedná. Unixové systémy používají 7 typů souborů – v tabulce 3 je jejich výpis, těm, jejich význam není teď jasný, se budeme věnovat později.

```
ls -la
total 20
drwxr-xr-x  5 justify justify 4096 Feb 23 15:58 .
drwxr-xr-x 26 justify justify 4096 Feb 23 21:40 ..
drwxr-xr-x  4 justify justify 4096 Feb 18 00:49 OS2
drwxr-xr-x  5 justify justify 4096 Dec  6 19:42 PS
drw-r-xr-x  2 justify justify 4096 Feb 23 16:02 pokusy

ls /home/justify/Dokumenty/ -la
total 20
drwxr-xr-x  5 justify justify 4096 Feb 23 15:58 .
drwxr-xr-x 26 justify justify 4096 Feb 23 21:40 ..
drwxr-xr-x  4 justify justify 4096 Feb 18 00:49 OS2
```

```
drwxr-xr-x  5 justify justify 4096 Dec  6 19:42 PS
drw-r-xr-x  2 justify justify 4096 Feb 23 16:02 pokusy
```

V prvních dvou příkladech je aktuální adresář právě adresář Dokumenty, v posledním případě na aktuálním adresáři nezáleží.

2.4.20 ln

Vytvoří odkaz na soubor („zástupce“). Pokud dojde ke smazání souboru, na který se odkazuje, odkaz zůstane zachová, ale bude rozbitý. Pokud pak soubor se stejným názvem znovu vytvoříme, odkaz bude opět funkční – pracuje tedy jen se jménem souboru. K odkazu mají všichni nastavena všechna přístupová práva.

```
ln -s copy.txt odkaz.txt

-rw-r--r-- 1 justify justify 0 Feb 23 16:02 copy.txt
lrwxrwxrwx 1 justify justify 8 Feb 23 15:59 odkaz.txt -> copy.txt
```

2.4.21 mkdir

Vytváří nový adresář. Jméno adresáře, popřípadě cesta, kde se má vytvořit, se předává jako parametr. Pokud není cesta specifikována, vytvoří se v aktuálním adresáři. Když se budeme snažit vytvořit adresář v cestě, která neexistuje, dojde k chybě. Pokud budeme chtít takovému chování zamezit, je třeba použít přepínač -p, který pak vytvoří všechny adresáře, které v zadané cestě neexistují.

```
mkdir test-dir
ls -la
...
drwxr-xr-x  2 justify justify 4096 Feb 23 22:04 test-dir
```

2.4.22 touch

Vytvoří nový prázdný soubor pokud je použit se jménem neexistujícího souboru. Pokud je jako parametr předán existující soubor, touch se souboru „dotkne“ čímž změní jeho timestamp – časová známka posledního přístupu k souboru.

```
touch text-file.txt
ls -la
...
-rw-r--r--  1 justify justify    0 Feb 23 22:07 text-file.txt
```

vytvoření nového souboru

```
ls -la
...
drwxr-xr-x  2 justify justify 4096 Feb 23 22:04 test-dir

touch test-dir
ls -la
...
-rw-r--r--  1 justify justify    0 Feb 23 22:07 text-file.txt
```

ukázka změny časové známky

2.4.23 du

Rekurzivně vypočítá (přesněji spíše odhadne) velikost adresářů v aktuálním adresáři. Volání je velice silně parametrizovatelné a tvar výstupu programu je tak možné silně ovlivnit.

2.4.24 file

Vypíše informace o souboru, který je předán jako vstupní parametr – respektive cesta k tomuto souboru.

```
file helloword.cpp
helloword.cpp: C source, ASCII text

file ~/data/tex/zivotopis.pdf
./data/tex/zivotopis.pdf: PDF document, version 1.2
```

2.4.25 cat

Vytiskne obsah souboru na standardní výstup. Parametr -n slouží pro očíslování vypisovaných řádků.

```
cat helloword.cpp
#include <iostream>

using namespace std;

int main() {
cout << "Hello Word!!" << endl;
return 0;
}
```

2.4.26 more a less

more je filtrovací program na interaktivní procházení souborů po stránkách, obrazovku po obrazovce. less je v manuálových stránkách uvedený jako „opposite of more“, je to ale program podobný more jen nabízí více možností. Další výhodou je například to, že během čtení nedochází k načtení celého souboru, což se u velkých souborů projeví rychlejším spuštěním.

2.4.27 head a tail

Zobrazí začátek souboru respektive jeho konec. Jako argument předáme vypisovaný soubor a taky číselný parametr, který určuje kolik řádku od začátku, respektive od konce, se má zobrazit (pokud není uvedeno, zobrazí 10 řádků).

```
head -n 1 helloword.cpp
#include <iostream>
```

Pokud je předané číslo záporné, znamená to, kolik řádků od konce se má vynechat.

```
head -n -7 helloword.cpp
#include <iostream>
```

Soubor má celkem 8 řádků, pokud jich 7 od konce odebereme, vytiskne se jen první řádek.

```
tail -n 1 helloword.cpp
}
```

Podobně jako u head můžeme i u tail specifikovat spíš než kolik řádků se má vytisknout, kolik se jich má od začátku vynechat. Tentokrát se před číslem musí objevit +.

```
tail -n +8 helloword.cpp
}
```

Můžete si všimnou, že ač má soubor právě 8 řádků, předchozí příkaz ještě vypíše poslední řádek. Je to proto, že tail začíná pracovat na k-tém řádku – v našem případě na osmém řádku.

2.4.28 wc

Zobrazí informace o souboru – počet řádků, slov a bajtů v tomto pořadí.

```
wc helloword.cpp
8  17 104 helloword.cpp
```

2.4.29 cp

Slouží ke kopírování souborů, adresářů. V případě kopírování neprázdných adresářů je třeba uvést parametr -r pro rekurzivní kopírování.

2.4.30 mv

Přesouvá soubory a adresáře - zároveň se používá pro přejmenovávání souborů (soubor přesuneme a umístíme pod novým jménem). Parametr -i pro interaktivní chování, -f pro nucený přístup.

2.4.31 rm

Mazání souborů a adresářů. U neprázdný adresářů je opět nutné uvést parametr -r pro rekurzivní přístup. Pozor, operační systém neposkytuje nástroje pro obnovu smazaných souborů. Příkaz rm bývá nastavený na takzvané bezpečné mazání. To znamená, že terminál bude vyžadovat potvrzení, zda chcete soubor (adresář) opravdu smazat. Toto chování lze ovlivnit pomocí parametru -i (pro interaktivní chování a -f (forced–nucené mazání bez dalších otázek).

```
rm -i copy.txt
rm: remove regular empty file ‘copy.txt’?
```

2.4.32 find

Surové vyhledávání souborů na disku s obrovským množstvím nastavení (-name, -regex, -size, -type, -perm, -mmin, -mtime, -delete), která buďto ovlivňují samotné hledání nebo stanovují akce, které se provedou po nalezení souboru.

2.4.33 locate

Další vyhledávání souborů, které pracuje se cache souborů, takže je rychlejší.

2.4.34 chmod

Změna přístupových práv (viz. níže).

3 Přístupová práva v Unixu

V Unixu je každému souboru nebo adresáři přidělen uživatel, který je jeho vlastníkem a skupina, do které spadá. Přístupová práva se pak nastavují zvlášť pro vlastníka (u – user), skupinu (g – group) a ostatní (o – others). Souborům pro jednotlivé účastníky lze přidělit právo na čtení (r – read), zápis (w – write) a spouštění (x – execute). Pro adresáře pak právo na výpis podadresářů a jejich souborů (r), vytváření podadresářů a souborů (w) a vstup do adresáře (x). Pro vyznačení se používají speciální bity, ve kterých jsou přístupová práva zapsána.

Přístupová práva se mění příkazem **chmod** ke kterému můžeme přidat parametr -R, který způsobí rekurzivní aplikaci změny (pokud měníme práva pro adresář, změní se stejně práva i pro všechny soubory a adresáře, které v něm jsou). Další věc, kterou můžeme u souborů měnit je skupina, do které soubor patří. K tomu slouží příkaz **chgrp**

Uživatel je v Linuxu jednoznačně identifikován svým UID – User Identification. Root má UDI rovno 0. Uživatelská jména se zavádí pouze pro zlepšení orientace uživatele při práci se systémem. V systému lze nají soubory, které nemají nastaveného žádného vlastníka. Ke „vzniku“ takových souborů dojde při odstranění záznamu o uživateli z databáze účtů, ale nejsou smazána jeho data, například z domovských adresářů.

Skupiny jsou podobně jako uživatelé identifikovány svým GID – Group Identification. Zopakujme, že informace o uživatelských účtech, skupinách a heslech jsou uloženy v /etc/passwd a /etc/group – tyto soubory mají podobnou strukturu. Jeden řádek v /etc/passwd vypadá takto:

```
<název účtu> : <heslo> : <UID> : <GID> : <jméno> : <adresář> : <shell>
```

V linuxu se záznamy oddělené dvojtečkou vyskytují poměrně často, převážně v konfiguračních a systémových souborech. Pokud položku **heslo** nastavíme na jeden znak, například „*“, zamezíme uživateli přihlášení do systému pomocí hesla. Předposlední údaj představuje domovský adresář daného uživatele a poslední údaj je implicitní shell – například /bin/bash. Význam ostatních údajů je zřejmý.

```
group novakj
students wheel
ls -l soubor1
-rwxr-xr-x 1 novakj student 69 Jun 18 16:45 soubor1
chgrp wheel soubor1
ls-l soubor1
-rwxr-xr-x 1 novakj wheel 69 Jun 18 16:45 soubor1
```

Uživatel může souboru nastavit příslušnost pouze do skupin, ve kterých je sám členem.

3.1 Zadání práv symbolicky

Práva zadáme v tomto tvaru [ugoa][+-=][rwxXstugo]. To znamená, že nejprve zadáme, koho se změna týká (a – all), poté uvedeme zda práva přidáváme, odebíráme a nebo „natvrdo“ nastavujeme a poté specifikujeme o jaká práva se jedná.

```
chmod u-w text.txt
```

Uživateli (u – user) odeber (-) práva na zápis (w – write) u souboru text.txt.

```
chmod u+rw text.txt
```

Uživateli (u – user) přidej (+) práva na čtení (r – read) a zápis (w – write) u souboru text.txt.

```
chmod a=rw, o= text.txt
```

Všem (a – user, group, others) nastav (=) práva na čtení (r) a zápis (w) a poté ostatním (o – others) nastav (=) doslova žádná práva (za rovná se je v příkazu mezera) u souboru text.txt.

```
chmod a - rwx, u=rwx mail
```

Všem odeber práva na čtení, zápis a spouštění a posléze uživateli nastav práva na čtení, zápis a spuštění.

3.2 Zadání práv osmičkově

Abychom mohli přístupová práva zadávat pomocí osmičkového zápisu, musíme si nejprve stanovit, jaké „hodnoty“ budou reprezentovat jaká práva. Pro čtení máme hodnotu 4 (4=r), pro zápis 2 (w=2) a pro spouštění hodnotu 1 (x=1). Práva nastavujeme pro všechny 3 skupiny současně a tak například hodnota 660 představuje práva na čtení a zápis pro uživatele a skupinu a žádná práva pro ostatní.

```
chmod 755 text.txt
```

Uživateli nastav plná práva (čtení, zápis, spuštění), skupině a ostatním práva pro čtení a spuštění.

4 Adresářová struktura systému Unix

Adresářová struktura v systémech unixového typu je jedna velká stromová struktura začínající v kořenovém adresáři. Soubory, které začínají tečkou jsou skryté a nejsou viditelné, pokud o ně nepožádáme. Tato struktura obsahuje některé velice důležité adresáře s pevně přiděleným účelem, u kterých je dobré vědět, co obsahují.

Z faktu, že je adresářová struktura reprezentovaná stromovou strukturou, plyne skutečnost, že každý uzel (kromě kořene) musí mít svého předka. Každý nelistový uzel je adresář a každý list je soubor. Ani prázdný adresář nebude list, protože i ten obsahuje dva záhadné adresáře (můžete ověřit pomocí prvního písmene výpisu ls -la), které jsou pojmenované „.“ a „..“ (tečka a dvě tečky). Ty vznikají spolu s adresářem během jeho vytváření. Díky tomu nedojde k tomu, že by byl nově vytvořený adresář prázdný. Adresář „.“ slouží jako odkaz na aktuální adresář a „..“ slouží jako odkaz na rodičovský adresář (předem ve stromové hierarchii).

4.1 Důležité adresáře

V tabulce 4.1 najdete seznam důležitých adresářů, které se v linuxových operačních systémech vyskytují spolu s popisem, co v nich je, k čemu slouží a co by se do nich mělo ukládat.

4.2 Expanze jmen

Expanze je soubor operací, které se s námi zadaným příkazem vykonají, než se spustí samotné jeho vykonávání. Může jít o rozepsání adres, substituci některých částí nebo rozšíření znaků, které pracují jako „syntaktický cukr“.

4.2.1 ~ tilda expanze

Pokud uvedeme ~ jako součást cesty, respektive její prefix, dojde k nahrazení za cestu od domovského adresáře aktuálně přihlášeného uživatele.

```
~/bin
```

se rozšíří na

```
/home/justify/bin
```

4.2.2 Doplnění cest * a ?

Jedná se o klasické doplňování jmen souborů, kde otazník (?) nahrazuje právě jeden znak a hvězdička (*) libovolný počet znaků (včetně žádného znaku). Samotná * tedy znamená všechny soubory v daném adresáři, *.* pak všechny soubory, které mají ve jméně alespoň jednu tečku.

/bin/	základní spustitelné systémové programy pro použití všemi uživateli
/boot/	jádro systému (kernel), soubory zavaděče
/dev/	soubory reprezentující fyzická zařízení nebo pseudozařízení systému
/etc/	globální konfigurační soubory systému
/home/	domovské adresáře uživatelů
/lib/	základní sdílené knihovny systému
/media/	obsah výměnných zařízení
/mnt/	ostatní souborové systémy, které nemusí být nutně výměnná zařízení (floppy, cdrom)
/opt/	softwarové aplikace, které nejsou standardní součástí distribuce
/proc/	soubory nastavení a stavu systému a jednotlivých procesů – „mapa stavu paměti RAM“
/proc/kcore	obraz celé virtuální paměti systému
/root/	domovský adresář superuživatele
/sbin/	systémové privilegované spustitelné soubory – programy potřebné pro zavedení a administraci systému – přístupné pro superuživatele root
/srv/	serverové aplikace
/sys/	virtuální systémový adresář
/tmp/	smetiště, dočasný adresář, vytvořit zde adresář může kdokoliv, ale smazat ho může jen vlastník
/usr/	obsahuje další stromovou strukturu, který obsahuje velké množství informací o systému, jeho nastavení, jeho zdrojové kódy, spustitelné a konfigurační soubory – veškerá data systému
/var/	soubory, jejichž obsah se během chodu systému dynamicky mění, například soubory s uživatelskou poštou, www stránky, archivy Network News, fronta pro tiskárnu a podobně
/dev/mem/	speciální soubor pro práci s pamětí
/dev/null	„černá díra“, bezedný kontejner, do kterého můžeme převést výstup, který nás nezajímá, zapsaná data nelze dostat zpět
/dev/full	simuluje zaplněné zařízení – testování výjimky zápisu do plného souboru
/etc/passwd/	soubor založených uživatelů
/etd/shadow	soubor uživatelských hesel a jejich vlastností (šifrované)
/etc/fstab/	(FileSystem TABle) popisuje jednotlivé systémové svazky připojitelné během života systému, obsah souboru hraje klíčovou roli při připojování souborových systému při startu systému

Tabulka 4: Popis důležitých adresářů v systému

4.2.3 Výběr z množiny []

Pomocí hranatých závorek [ ] můžeme zadat více variant pro určitou pozici znaku.

```
/home/m[ai]rek
```

tedy znamená současně

```
/home/mirek
```

a

```
/home/marek
```

5 Systém procesů

Program je spustitelný soubor uložený na disku. Je-li program spuštěn, jádro zavede jeho obraz do paměti a začne vykonávat v něm obsažené instrukce. Proces je aktivní reprezentace programu. Každý proces při svém vzniku získá PID – unikátní číselný identifikátor procesu. Plánovač procesů procesům cyklicky přiděluje výpočetní čas procesoru a umožňuje tak multitasking. Stejně jako adresářová struktura je i systém procesů stromová hierarchie. Kořenový proces se jmenuje init a má PID 1. Je to proces, který běží vždy, když běžím systém. Každý proces musí mít nastaveného předka (rodiče).

Jako shell se označuje příkazový interpret, který vykonává příkazy v textovém režimu. Stará se o jejich spouštění a správu – vykonávaným příkazům se pak říká úlohy.

Během práce se systémem se lze setkat se třemi druhy procesů:

- Uživatelské procesy – userspace processes – jsou spouštěny jednotlivými uživateli.
- Systémové procesy – deamons – jsou spouštěny během startu systému, zpravidla v nekonečné smyčce a zajišťují různé služby jako třeba periodickou obsluhu klientů.
- Vlákna jádra – kernel threads – jsou v podstatě speciální části jádra běžící jako procesy. Například vlákno **kmod** se stará o zavádění vyžadovaných modulů jádra do paměti.

5.1 Výpis procesů

5.1.1 ps aux

Vypíše všechny běžící procesy na standardní výstup.

5.1.2 top

Jak už název napovídá, toto zobrazení je nejlepší. Výpis v hlavičce obsahuje statistické informace o celkovém množství běžících procesů, kolik z nich aktuálně běží, kolik jich je zastavené, spí nebo je zombie. Dále zobrazuje souhrnné informace o operační paměti a procesoru. Tabulka procesů je seřazená podle % využití procesoru, navíc se sama po několika vteřinách aktualizuje. Jak už je obvyklé, náhled programu ukončujeme klávesou **q**. Pro zobrazení nápovědy lze použít klávesu **h**

5.1.3 pstree

Zobrazení stromové ASCII art hierarchie procesů.

5.2 Ukončení procesu

5.2.1 kill

Program **kill** funguje tak, že procesu pošle číselný signál a to ve tvaru:

```
kill -<signál> <procesy>
```

Za signál můžeme dosadit buďto číslo nebo jméno (konstantu) signálu. K některým signálům lze implementovat jejich zpracování v programu, některým se ale nelze bránit. K bezprostřednímu ukončení procesu zle použít signál 9 (SIGKILL, který spolu s SIGSTOP patří mezi signály, kterým se proces nedokáže bránit). Jako další signály můžeme uvést SIGSTOP a SIGCOUNT díky kterým můžeme proces pozastavit a poté ho třeba přesunout na pozadí. Za parametr procesy můžeme dosadit buďto PID procesu nebo identifikátor procesu v rámci aktuálního shellu, ve tvaru %<id>. V příkladu číslo 544 reprezentuje PID ukončovaného procesu.

```
kill -9 544
```

Informace o signálech, které je možné používat a k čemu lze nalézt v manuálových stránkách pomocí příkazu:

```
man 7 signal
```

Při ukončení proces vrací svému rodiči číselnou hodnotu. Konvence a zvyklosti stanovují, že pokud program vrátí hodnotu 0, jeho výpočet skončil v pořádku. Jakákoliv nenulová hodnota znamená chybu. Při ukončení procesu jsou ukončeny i jeho potomci – rodič má zodpovědnost za svoje potomky, tedy i za jejich korektní ukončení. Rodič může zařídit, aby nebyli při jeho smrti ukončeni jeho potomci. Pokud je rodič ukončen a jeho potomci zůstávají, nastaví se jejich rodič (protože každý proces musí mít nastaveného rodiče) na proces init (PID 1).

5.3 Plánované spouštění programů

V unixových operačních systémech je možné plánovat spouštění programů a vykonávání příkazů. Jinak řečeno, uživatel může stanovit čas a proces, který má být v daný čas spuštěn. O takové spuštění se pak nestará jádro operačního systému, ale je plně v režii systémových deamonů. Plánované spuštění může být buďto jednorázové nebo periodické.

5.3.1 at

Slouží k jednorázovému plánovanému spuštění procesu. Jako parametr se předá jméno souboru, který se ve stanovenou dobu spustí. Program lze použít ve tvaru:

```
at <čas> <datum> +<přírustek>
```

Příkaz v tomto tvaru spustí interaktivní režim, ve kterém je ještě potřeba zadat program, který bude spuštěn.

```
at 11:30 26.4.2015
warning: commands will be executed using /bin/sh
at> ls -la ~
at> C-d
job 3957 at Wed Mar 11 20:58:00 2015
```

Číslo job nelze chápat jako PID, protože proces v době plánování ještě neběží, je pouze vytvořen záznam o jeho spuštění. Seznam naplánovaných procesů můžeme vypsat pomocí příkazu **atq**.

```
atq
3957    Wed Mar 11 20:58:00 2015 a betajo00
```

Naplánované procesy lze zrušit programem **atrm**, kterému jako parametr předáme id naplánované úlohy.

```
atrm 3957
```

5.3.2 cron a crontab

Podobně jako **at** umožňuje **cron**, respektive **crontab**, spustit plánovaný proces ale s tím rozdílem, že umožňuje periodické spouštění (i **at** umožňuje periodické spouštění a to v případě, že poslední předaný příkaz bude opětovné spuštění sebe sama – v takovém případě je ale nutné použít časový posun). O spuštění programu **crontab** se staré deamon **bf**. Program **crontab** bývá používán dvěma způsoby:

```
crontab -u <uživatel> <soubor>
crontab -l <uživatel>
```

První příkaz nastaví tabulku spouštěných procesů pro uživatele <uživatel>, přitom vzorem tabulky je soubor <soubor>. Druhým příkazem je možné zobrazit nastavenou tabulku. Každý řádek v tabulce reprezentuje jednu periodicky spouštěnou úlohu. Začíná-li řádek znakem hash # („kanál“), pak se jedná o komentář a celý řádek je ignorován. Ostatní řádky musí mít tvar:

```
<minuta> <den> <měsíc> <den v týdnu> <příkaz>
```

Časové údaje mohou být číselné nebo místo nich může být uvedený znak *, který znamená „kdykoliv“. Číselný údaj může být uveden pouze jeden nebo musí být odděleny čárkami – to v případě, že bychom chtěli script spouštět například ve čtvrtek i v pátek. V následujícím příkladu je použit příkat **cat**, aby byla očividná struktura uložení procesů.

```
cat ~/cron-my.tab
#moje akce
5 0 * * * ~/prvni-script
0, 10, 20, 30, 45, 50 14 1 * *  ~/druhy-script
15 4 * * sun ~/treti-script
```

Pozor! Pokud nastavíme hvězdičky u všech časových parametrů, bude se script spouštět každou minutu až do konce věků druidů.

5.4 Priority procesů

Procesům se nastavuje priorita v intervalu od -20 (nejvíce agresivní) do 19 (nejméně agresivní). Priorita procesům určitě, jak často jim bude přidělován procesorový čas. Uživatel může nastavovat, respektive měnit, prioritu pouze svým procesům.

5.4.1 nice

Příkaz nice slouží ke změně priority neběžících procesů – spustí proces s předanou prioritou

```
nice --5 bluefish
```

Spustí editor Bluefish s prioritou -5. Základní priorita je nastavena na 10. Zápornou prioritu ale může nastavovat pouze root.

5.4.2 renice

Slouží ke změně priority již běžících procesů.

```
nice -n 10 ~/data/archivuj &
[1] 24532
renice 20 -p 24532
24532: old priority 10, new  priority 20
```

5.5 Řízení úloh v shellu

Úlohy můžou běžet na pozadí nebo na popředí. Pokud běží úloha na popředí, blokuje shell, do kterého pak není možné zadávat další příkazy – programu je umožněn výstup na terminál. Pokud běží na pozadí (toho docílíme tak, že při spouštění uvedeme na konec &), je možné zadávat další příkazy a spouštět další programy. Program běžící na pozadí by ale neměl vypisovat výstup na terminál – to by bylo velice rušivé. Zároveň by neměl být interaktivní. Interakce s uživatelem by opět způsobila zásah do okna shellu, tedy zásah do aktuální práce uživatele.

```
$~/data/archivuj &
[1] 15635
```

... číslo v závorkách je identifikátor procesu v rámci aktuálního shellu, druhé číslo je PID procesu. Protože je program spuštěn pomocí shellu, bude po jeho úspěšném ukončení vypsáno:

```
[1]+ Done ~/data/archivuj
```

Respektive:

```
[1]+ Exit 10 ~/data/archivuj
```

v případě, že byl program ukončen s jinou návratovou hodnotou, než je 0. Tato návratová hodnota je pak vypsána za klíčovým slovem exit.

K výpisu aktuálně běžících úloh slouží příkaz jobs. Pro převedení úlohy z pozadí na popředí můžeme použít příkaz **fg**. Při opačném převodu – z popředí na pozadí – musíme použít vícekrokový přístup. Úlohu nejprve pomocí CTRL + Z pozastavíme. Teprve poté můžeme pomocí příkazu **bg** úlohu převést na pozadí. Proč bychom chtěli nějakou úlohu provádět na pozadí? Unix je ze své podstaty víceúlohový a na pozadí, přesně jak bychom očekávali, probíhá velké množství systémových úloh bez našeho aktivního vědomí. Přidat k nim (na pozadí) další úlohu může být výhodné ve chvíli, kdy nás námi spuštěná úloha dobou svého zpracování nutí si jít uvařit patnáctou kávu (třeba dlouhé rozbalování programem **gzip**. Pro doplnění a zopakování uvedeme klávesovou zkratku CTRL + C, která provede násilné ukončení úlohy běžící na popředí.

Jako zombie se označuje proces, u kterého si rodič ještě nevyzvedl návratovou hodnotu. Název zombie je poměrně vystihující, protože proces pošle informaci o svém ukončení pomocí signálu SGCHLD a poté se ukončí. Jádro si poté informace o procesu udržuje do té doby, než si jeho návratovou hodnotu jeho rodič nevyzvedne, a to pomocí volání jádra wait().

6 Vstup a výstup programu

Každý program má vstup, výstup a chybový výstup pro výstup chyb. Program tak můžeme chápat jako filtr, který převede vstup na výstup. Vstup souboru můžeme přesměrovat tak, aby jím byl soubor.

```
program < soubor
```

Naopak můžeme i výstup programu přesměrovat do souboru. Protože se výstup dělí na výstup a chybový výstup, můžeme zvolit, který z nich chceme přesměrovat. Pro výstup programu použijeme >, pro chybový výstup 2> a pro veškerý výstup použijeme &>

```
program > soubor
program 2> soubor
program &> soubor
```

Výstup jednoho programu můžeme také přesměrovat na vstup jiného programu pomocí roury |.

```
program1 | program2
```

Výstup programu program1 se stane vstupem programu program2.

Pojmenovaná roura (princip fifo) je speciální soubor, který reprezentuje rouru. Pomocí ní je možné přesměrovat výstup z nějaké programu do tohoto speciálního souboru – do roury – a pak jiným programem tuto rouru číst, takže vstup programu se přesměrovává z roury.

-f	ignoruje velikost písmen
-n	číselné třídění (numeric)
-r	reverse – obrácené, sestupné třídění
-u	duplicitní řádky vypíše pouze jednou
-k	umožní výběr sloupce, podle kterého se třídí

Tabulka 5: Přepínače programu sort

7 Základy programování v shellu

Shell je plnohodnotný programovací jazyk. Umí pracovat s proměnnými, zpracovávat podmínky, provádět cykly, vyhodnocovat celočíselnou (pouze) aritmetiku, vytváře nové funkce a vytvářet výstup. Jako skript se v shellu označuje soubor s programem.

(jednoduché) programy (filtry, vstup/výstup)
+
roura
+
programovací jazyk
=
cokoliv!

[Out12]

7.1 Základní zpracování textu

Pro zpracování textu máme k dispozici již představené a z velké většiny jednoúčelové programy text, head, tail, wc, sort, cut, paste, diff. . . . Soubory v unixových systémech jsou tradičně textové, kde řádek je posloupnost tisknutelných znaků zakončených znakem ukončení řádku – end of line, což je zde typicky \n.

S příkazy cat, head a wc jsme se dostatečně seznámili v kapitole 2.4 a teď už se budeme věnovat pouze jejich aplikaci. K doplnění ještě uvedeme příkaz **tac**, který funguje stejně jako cat, jen vypisuje řádky od konce souboru.

7.1.1 Třídění řádků a sloupců souboru sort

Na textový soubor lze snadno pohlížet i jako na tabulku, kde je potřeba jeden znak vybrat jako oddělovač sloupců. Program sort projde řádky souboru a podle zadaných parametrů je setřídí. Pokud nemá, kromě tříděného souboru, zadaný žádný parametr, třídí lexikograficky vzestupně. Třídění pak můžeme ovlivnit přepínači dle tabulky 7.1.1.

7.1.2 Práce se sloupci cut a paste

Program cut umožňuje ze souboru „vyříznout“ vybraný sloupec. Výchozím oddělovačem, který cut používá pro rozdělení souboru do sloupců, je tabulátor. Pomocí parametru **-d** můžeme nastavit libovolný oddělovač. Parametr **-f** nám dle čísla (zleva se sloupce „číslují“ od 1) určit sloupec, který vybíráme.

```
cat tab2.txt
1 a 1 a
2 b 2 b

cut -f 1 tab2.txt
1
2
```

Program paste nám umožní vyříznuté sloupce souborů opět spojit do jednoho. Přepínač **-d** určuje, jaký oddělovač se má použít pro spojení sloupců. Pokud budeme chtít spojit sloupce pod sebe, nikoliv vedle sebe, můžeme použít soubor cat s více vstupními soubory.

7.1.3 Rozdíl souborů a adresářů diff

K určování rozdílu mezi dvěma soubory slouží program **diff**. Program diff vypíše výstup, který popisuje rozdíly mezi soubory po řádcích.

Reference

[Vych03] Vilém Vychodil: *Linux: Příručka českého uživatele*. Computer Press 2003, Brno ISBN 80-7226-333-1.

[Out12] Jan Outrata: *Základy systému UNIX a GNU/Linux*. Univerzita Palackého v Olomouci 2012.