

# Cvičení 3.

**ZOS 2014**

---



# Set EUID bit

- obvykle běží proces s **právy uživatele**, který jej spustil
- někdy to však nestačí...
- program `/usr/bin/passwd` by například potřeboval změnit hash hesla v souboru `/etc/shadow`, kam běžný uživatel nemá přístup
- jak je to zařízené?

```
ls -l /usr/bin/passwd
```

```
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
```

- vlastník má nastavený Set EUID bit (s)
- program bude spuštěn s právy vlastníka (root), nikoliv s právy běžného uživatele (!)



# ukázka

```
eryxl> ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 31704 2009-11-14 15:41 /usr/bin/passwd
eryxl>
eryxl> file /usr/bin/passwd
/usr/bin/passwd: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV)
, dynamically linked (uses shared libs), for GNU/Linux 2.6.8, stripped
eryxl> 
```



# Nastavení přístupových práv

- `chmod ug+rwX soubor`
- Práva lze zadat i číselně
  - r,w,x .. Tři trojice
  - Binárně 000 až 111 .. 0 až 7 .. osmičková soustava
- Příklady
  - `chmod 777 ahoj.txt`                      - **rwX** rwX rwX
  - `chmod 700 ahoj.txt`                      - rwX --- ---
  - `chmod 644 ahoj.txt`                      - rw- r-- r--



# Přístupová práva - pokus

- Pokusy vždy v nějakém pracovním adresáři:  
mkdir zos ; cd zos
  - touch ahoj.txt ; ls -l; chmod 777 ahoj.txt ; ls -l
- 
- |              |           |       |                         |
|--------------|-----------|-------|-------------------------|
| □ -rw-----   | 1 pesicka | users | 0 Oct 12 10:15 ahoj.txt |
| □ -rwxrwxrwx | 1 pesicka | users | 0 Oct 12 10:15 ahoj.txt |

# Přístupová práva

u (user)	g (group)	o (others)	u+g+o=a (all)
vlastník	skupina	ostatní	
- - -	- - -	- - -	
r w x	r w x	r w x	(read, write, execute)
- r w - 4+2= <u>6</u>	r - - <u>4</u>	- - - <u>0</u>	s1.txt => chmod <u>640</u> s1.txt

ls -l s1.txt  
1. sloupec:

- obyčejný soubor
- d adresář
- b blokové zařízení
- c znakové zařízení
- l symbolický link



# umask

- Maska přístupových práv při vytváření souborů
- Obsahuje práva, která „vypne“ (doplněk)
- Samotný **umask** – vypíše aktuální nastavení
  - Např. `umask ->` vypíše 77
  - Tj. group, other nebudou mít žádná práva při vytvoření nového souboru



# Umask - příklad

□ umask 007 ; touch nazdar1 ; ls -l

□ umask 000 ; touch nazdar2 ; ls -l

□ umask 077 ; touch nazdar3 ; ls -l

□ -rw-rw---- 1 pesicka users 0 Oct 12 10:42 nazdar1

□ -rw-rw-rw- 1 pesicka users 0 Oct 12 10:42 nazdar2

□ -rw----- 1 pesicka users 0 Oct 12 10:43 nazdar3





# Symbolický link, hard link

- `ln -s stare_jmeno nove_jmeno`
    - Symbolický
    - Smazání linku nesmaže soubor
    - Smazání souboru – broken link
    - Použijí se přístupová práva k souboru, nikoliv k linku
    - Často používáné, viz např. na eryxu `ls -l /`
  - `ln stare_jmeno nove_jmeno`
    - Hard link, pevný odkaz
    - Zvyšuje počet referencí na soubor
    - Staré i nové jméno jsou naprosto rovnocenné
-



# symbolický a hardlink

```
eryx1> ls -l
total 1
-rw----- 1 pesicka users 14 2012-10-10 09:23 s1.txt
eryx1>
eryx1> ln -s s1.txt symbol_link
eryx1>
eryx1> ln s1.txt hard_link.txt
eryx1>
eryx1> ls -l
total 3
-rw----- 2 pesicka users 14 2012-10-10 09:23 hard_link.txt
lrwxr-xr-x 1 pesicka users 6 2012-10-10 09:24 symbol_link -> s1.txt
-rw----- 2 pesicka users 14 2012-10-10 09:23 s1.txt
eryx1>
```



# Link – příklad

```
vi s99.txt
```

```
ln -s s99.txt mujlink
```

```
ls -l ; cat mujlink
```

```
vi s98.txt
```

```
ln s98.txt mujhardlink
```

```
ls -l ; {všimněte si počtu odkazů}
```

```
rm muj*link ; ls -l {smaže linky, sníží počítadlo s98}
```

který z linků mění počet odkazů na soubor?



# Možnosti vytvoření souboru

□ **mcedit** soubor.txt

□ **touch** soubor.txt

□ **cat** > soubor.txt (ukončení **Ctrl+D**)

□ **vi** soubor.txt

---



# Editor vi

**vi** soubor.txt (vim)

- a .. vkládání
- Píšeme text
- Esc :
  - wq .. Uloží a ukončí (write, quit)
  - q! .. Ukončí bez uložení změn
  - syn on .. Zvýraznění syntaxe

každý musí umět základní  
editaci souboru pomocí vi, vim



# Výpis textového souboru

- ❑ `cat s1.txt`
- ❑ `cat s1.txt s2.txt s3.txt`
- ❑ `cat s1.txt s2.txt s3.txt > celek.txt`
- ❑ `more s1.txt`
- ❑ `less s1.txt`
  
- ❑ `cat telefony.txt | grep marenka`
  - Vypíše řádky, obsahující string „marenka“



# Kolony příkazů

- Výstup příkazu 1 slouží jako vstup příkazu 2
- Pro spojení příkazů do kolony se používá |

```
cat /etc/passwd | grep pesicka | wc -l
```

- **POZOR !! NEPLÉST NÁSLEDUJÍCÍ**
- *Příkaz1 | příkaz2*
- *Příkaz1 > soubor*



# Příklad – častá chyba (!)

## □ *cat neco.txt | head*

- Příkaz *cat* pošle jednotlivé řádky souboru *neco.txt* dalšímu příkazu *head*
- Příkaz *head* vypíše prvních 10 řádek z dat, která dostal

## □ *cat něco.txt > head*

- Výstup příkazu *cat* je přesměrován do **souboru** *head*
- Bude vytvořen soubor *head* v aktuálním adresáři obsahující stejné řádky souboru jako soubor *neco.txt*





# Přesměrování do souboru

- Používejte `/bin/bash`
  - `cat /etc/passwd | head > vystup.txt`
  - `cat neco.txt > vysledek.txt`
    - Pokud `vysledek.txt` něco obsahoval, bude jeho obsah **přepsán**
  - `cat neco.txt >> vysledek.txt`
    - Připojí **na konec** souboru `vysledek.txt`
-



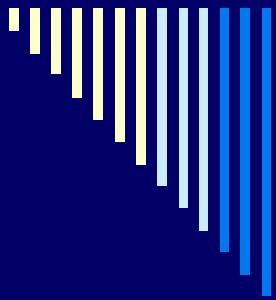
# Standardní a chybový výstup

- `./program > vystup.txt 2> chyby.txt`
  - Standardní výstup půjde do souboru *vystup.txt*
  - Standardní **chybový** výstup půjde do *chyby.txt*
  
- `more < vstup.txt`
  - Bude zpracovávat vstup ze souboru *vstup.txt*
  - (Pozn. u `more` lze i jen: `more vstup.txt`)



# File deskripty

- 0 .. st.vstup, 1 .. st.výstup, 2 .. st.chybový výstup
- ls > vystup.txt
- ls 1> vystup.txt
- ls 2> chyby.txt
- ls 1>vse.txt 2>&1
  - Napřed přesměruje deskriptor 1, bude ukazovat na *vse.txt*
  - Poté přesměruje deskriptor 2, bude ukazovat také na *vse.txt*
- ls 2>&1 1>vse.txt .. Funguje jinak, proč?
- ls 1>>vse.txt 2>>vse.txt



# Procesy

eryx4> **ps x**

PID	TTY	STAT	TIME	COMMAND
15055	pts/6	S	0:00	-tcsh
15256	pts/6	R	0:00	ps x

eryx4> **top &**

[1] 15260

eryx4>

[1] + Suspended (tty output) top



# Procesy 2.

eryx4> **ps x**

PID	TTY	STAT	TIME	COMMAND
15055	pts/6	S	0:00	-tcsh
15260	pts/6	<b>T</b>	0:00	top .. ma Tcko
15261	pts/6	R	0:00	ps x

PS – **man ps**

## PROCESS STATE CODES

- D uninterruptible sleep (usually I/O)
- R runnable (on run queue)
- S sleeping
- T** **traced or stopped**
- Z a defunct ("zombie") process



# Procesy 3.

eryx4> **jobs**

[1] + Suspended (tty output) top

eryx4> **jobs -l**

[1] + 15260 Suspended (tty output) top

**fg** .. proces na popředí

**fg** *cislo* .. vybereme, který na popředí

**kill** *cislo\_procesu* ; **xkill** .. pošle signál TERM procesu

**kill -9** *cislo\_procesu* .. pošle signál 9 procesu

**pstree** .. strom procesů

---



# Číslování řádků v souboru

❑ `cat /etc/passwd | nl | head`

- vypíše prvních 10 očíslovaných řádků

Ize i  
head -15

❑ `cat /etc/passwd | nl | head -n 15`

- vypíše prvních 15 očíslovaných řádků

poslední 3  
řádky

❑ `cat /etc/passwd | nl | tail`

- vypíše posledních 10 řádků

od 3. řádky  
do konce

❑ `cat /etc/passwd | nl | tail -n 3`

❑ `cat /etc/passwd | nl | tail -n +3 | more`



# Hledáme soubory s **find**

**find** /etc **-name** passwd **-print**

kde hledáme: /etc

co hledáme: soubor s názvem passwd

akce: vypíšeme nalezené soubory  
(-print, -exec, -ok)

☐ **find** /bin **-name** date **-print**

☐ **find** /dev **-type** c **-print**

☐ **find** /dev **-type** b **-print**

---





# find – příklady

□ **find** . -name "cit\*" -ok rm {} \;

- Prohledá aktuální adresář
- Najde soubory začínající na cit
- U každého se zeptá, zda jej může smazat (pokud by akce byla -exec, maže bez ptaní)

□ **find** \$HOME -mtime 0

- Soubory v dom. adresáři modifikované během posledních 24 hodin
  - (čas od modifikace je dělený 24 hod, tedy 0)
-



# Find - příklady

❑ `find . -perm 644`

- Hledá v akt. adresáři a podadres. soubory s právy 644

❑ `find . -perm -644`

- Jako předchozí, ale můžou být i práva navíc

❑ `find . -perm /222`

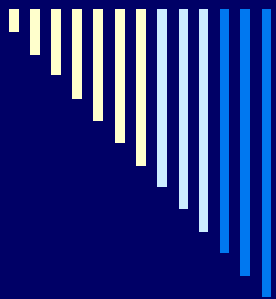
- Alespoň někdo musí mít právo zapisovat

❑ `find . -perm /u+w,g+w`

- Stejně jako předchozí příklad, v symbolickém zápisu

❑ `find /var/spool -mtime +60` .. Modifikace před > 60 dny

❑ `find . -type f -exec file '{}'` \; .. určete ☺



# Wildcards \* ? a další

❑ `cd /bin`

❑ `ls m*`

❑ `ls mk*`

0 či více znaků

❑ `ls m?`

právě jeden znak

❑ `ls m[a-u]`

jeden znak ze skupiny

❑ `ls *a[b-e]*`

❑ `ls m[k,o]*`

❑ `ls "*"c`

neinterpretuje obsah uvozovek

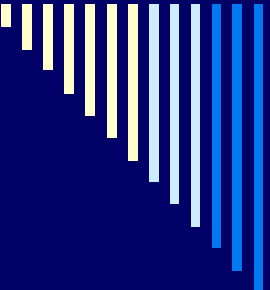


# Příklad 1 – uživatelé

□ who

<i>pesicka</i>	<i>pts/1</i>	<i>Oct</i>	<i>5</i>	<i>23:43</i>	<i>(84.242.95.197)</i>
<i>student6</i>	<i>ttyp0</i>	<i>Oct</i>	<i>6</i>	<i>00:21</i>	<i>(hyperochus.zcu.cz)</i>
<i>fhoudek</i>	<i>pts/3</i>	<i>Oct</i>	<i>6</i>	<i>00:09</i>	<i>(koleje-zcu.souep1.cz)</i>
<i>maskova</i>	<i>pts/6</i>	<i>Oct</i>	<i>5</i>	<i>20:25</i>	<i>(b1.sab.plz.sloane.cz)</i>
<i>student6</i>	<i>ttyp1</i>	<i>Sep</i>	<i>27</i>	<i>14:32</i>	<i>(hyperochus.zcu.cz)</i>

□ chceme seznam uživatelů, abecedně seřazený  
a bez duplikací

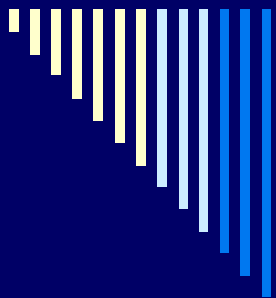


# Příklad 1 – uživatelé

□ `who | cut -c1-8`

```
pesicka  
student6  
fhoudek  
maskova  
student6
```

□ **vybere jen znaky 1 az 8**



# Příklad 1 – uživatelé

□ `who | cut -c1-8 | sort`

*fhoudek*

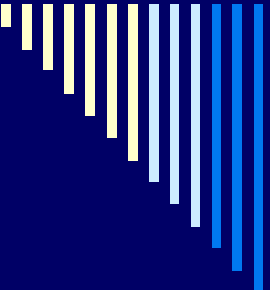
*maskova*

*pesicka*

*student6*

*student6*

□ **seřídí podle abecedy**



# Příklad 1 – uživatelé

□ `who | cut -c1-8 | sort | uniq`

*fhoudek*

*maskova*

*pesicka*

*student6*

□ odstraní duplicity seřazených řádků

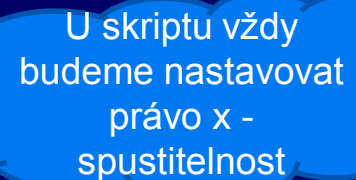


# Příklad 1 – první skript

□ **vi** uzivatele.sh

```
#!/bin/bash
```

```
who | cut -c1-8 | sort | uniq
```



U skriptu vždy  
budeme nastavovat  
právo x -  
spustitelnost

□ **chmod** +x uzivatele.sh

■ Důležité: skriptu nastavíme právo x !

□ **./**uzivatele.sh





# Příklad – interaktivní skript

```
#!/bin/bash
```

```
echo Jak se jmenujes?
```

```
read JMENO
```

```
echo Vitam uzivatele: $JMENO
```

```
echo Mas login: $USER
```

```
sleep 3
```

```
echo Vypisu ti kalendar
```

```
cal
```

---



# Poznámky

bez mezery !!!!!!!!!!!

- **#!/bin/bash** -vždy na 1.řádce našich skriptů
  - Jaký příkazový interpret se bude používat
- **read JMENO**
  - Načte vstup uživatele do proměnné JMENO
- **echo \$JMENO**
  - Přístup k proměnné JMENO – všimněte si \$
- **sleep 3** – pauza na 3 sekundy
- **cal** – kalendář, viz man cal



## příklad – skript s parametry

```
#!/bin/bash
```

```
echo Budu analyzovat soubor $1
```

```
echo Analyza souboru $1      > vystup.txt
```

```
echo -n "Provedeno dne "    >> vystup.txt
```

```
date    >> vystup.txt
```

```
file $1  >> vystup.txt
```

```
echo "Analyzu provedl: " >> vystup.txt
```

```
whoami >> vystup.txt
```

---

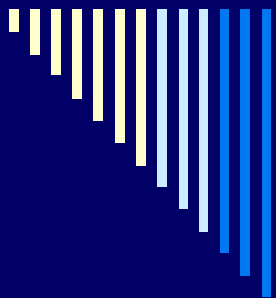


# Více adresářů najednou

**mkdir** ahoj

**mkdir** ahoj/{jedna,dve,tri}

- adresář ahoj vytvoří první příkaz
- druhý příkaz vytvoří všechny tři podadresáře



# Nahrazování slov

echo MaLa VELKA Pismena | tr '[A-Z]' '[a-z]'

vypíše:

mala velka pismena

znaky z množiny [A-Z] nahrazuje znaky [a-z]



# Příklad – četnost slov v anglickém textu

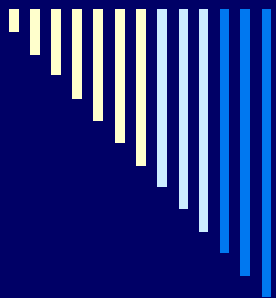
- Dále uvedený příklad na četnost slov je jen ilustrační, není třeba jej z hlavy sestavit ☺ stejně tak neošetřuje všechny možnosti
- Napřed ve vstupním souboru unifikuji velikost znaků na malá písmena
- `tr '[A-Z]' '[a-z]' < textik.txt`



## Příklad – četnost slov

Odstraníme interpunkci, slova oddělená  
mezerami

```
tr '[A-Z]' '[a-z]' < textik.txt |  
tr -cd '[A-Za-z0-9_ \012]'
```



## Příklad – četnost slov

každé slovo na novém řádku

```
tr '[A-Z]' '[a-z]' < textik.txt | tr -cd '[A-Za-z0-9_ \012]' |  
tr -s '[' '\012'
```





# Příklad – četnost slov

Výstup seříděný podle slov, odstraněné duplicity, u každého slova uvedeno, kolik má duplicit

```
tr '[A-Z]' '[a-z]' < textik.txt | tr -cd '[A-Za-z0-9_ \012]' |  
tr -s '[]' '\012' | sort | uniq -c
```

**uniq** se používá až na seříděném souboru (sort)  
**uniq -c** ... prefixuje řádky počtem výskytů řetězce



# Příklad – četnost slov

seřadit podle četnosti výskytu - finále

```
tr '[A-Z]' '[a-z]' < textik.txt | tr -cd '[A-Za-z0-9_ \012]' |  
tr -s '[' '\012' | sort | uniq -c | sort -nr
```

Konečně výsledný skript

Uveden je opravdu jen na ukázkou, aby bylo vidět, jaké  
funkcionality lze spojováním příkazů dosáhnout



# Filtr **tee**

- Kopíruje standardní vstup na std. výstup
- A **současně** zapisuje do uvedeného souboru
  - Např. kopírování mezivýsledků
  - Analogie – pipe (potrubí),  
tee (odbočka vodovodního řadu, téčko)

```
ls -l | tee soubor.txt | grep ahoj
```

Uloží výstup příkazu  
ls -l do souboru

Zároveň jde výstup ls -l na std. výstup,  
kde jej dále zpracuje příkaz grep



# Komprese souborů - gzip

## □ gzip

- Komprimuje zadaný soubor (LZ77 algoritmus)
- Původní soubor přestane existovat, je nahrazen komprimovaným !!!!

## □ gzip soubor.txt ; ls -l

```
-rw-----  1 pesicka  users      59 Oct 12 10:59 soubor.txt.gz
```

□ gunzip soubor.txt.gz .. dekomrimace, volá gzip -d

□ gzip -d soubor.txt.gz .. dekomprimace



# Komprese souborů – bzip2

- Jiný komprimační algoritmus
- Výsledný soubor menší x více zatěžuje CPU

- **bzip2 soubor.txt**

- -rw----- 1 pesicka users 72 Oct 12 10:59 soubor.txt.**bz2**

- **bzip2 -d soubor.txt.bz2**

- Stejně tak lze použít bunzip2



# tar - archivace

- ❑ původně pro archivaci na pásku
- ❑ Z několika souborů a adresářů – 1 velký soubor
- ❑ Distribuce sw v balíčku – tarball – .tar.gz
- ❑ Zabalení a následná komprimace
- ❑ tar.gz, tar.Z, tar.bz2
- ❑ Parametr -z před rozbalení použije gunzip



# Vytváření archivu

```
tar -cvzf archiv.tar.gz ./data
```

- Volba -c vytváří archiv
- Volba -v ukecaný (verbose)
- Volba -z komprese gzip (-j pro bzip2)
- Volba -f soubor
- Soubory z podadresáře *data* sbalí do archivu

Vytvořený archiv můžeme prohlédnout přes *mc*

---



# Rozbalení archivu

□ `tar -xvzf archiv.tar.gz`

□ Parametry

- `-x`      rozbalit (extract)
- `-v`      upovídaný (verbose)
- `-z`      nejprve použije gunzip  
(pro bzip2 by bylo `-j`)
- `-f`      následuje jméno souboru

□ Vybalené soubory ukládá do aktuálního adresáře,  
tj. zde např. vytvoří podadresář data

---





---

# Zip, unzip

- `zip archiv *`
  - Vytvoří archiv.zip
- `unzip archiv.zip`