

CVIČENÍ 3.

ZOS 2024

L. Pešička



SET EUID BIT

- proces běží (obvykle) 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> █
```

PŘÍSTUPOVÁ PRÁVA (OPAKOVÁNÍ)

- `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` - **rw**x **rw**x **rw**x
 - `chmod 700 ahoj.txt` - **rw**x --- ---
 - `chmod 644 ahoj.txt` - **rw**- **r**-- **r**--

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**
 - Představuje 077
 - Vlastníkovi nic nevypíná (0)
 - Group, other nebudou mít žádná práva při vytvoření nového souboru, neboť 7 znamená vypni vše

UMASK - PŘÍKLAD

- `umask 007 ; touch nazdar1 ; ls -l`
- `umask 000 ; touch nazdar2 ; ls -l`
- `umask 077 ; touch nazdar3 ; ls -l`

- | | | |
|---------------------------|------------------------------|-------------------------------------|
| ▪ <code>-rw-rw----</code> | <code>1 pesicka users</code> | <code>0 Oct 12 10:42 nazdar1</code> |
| ▪ <code>-rw-rw-rw-</code> | <code>1 pesicka users</code> | <code>0 Oct 12 10:42 nazdar2</code> |
| ▪ <code>-rw-----</code> | <code>1 pesicka users</code> | <code>0 Oct 12 10:43 nazdar3</code> |

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 používané, 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> █
```

PŘÍKLAD S LINKY

```
touch s92.txt
```

```
ln -s s92.txt mujlink
```

```
ls -li
```

```
touch s93.txt
```

```
ln s93.txt mujhardlink
```

```
ls -li
```

-> všimněte si počtu odkazů

který z linků mění
počet odkazů na
soubor?

Zapamatujte si příkaz
ls -li zobrazí
čísla i-uzlů

MOŽNOSTI VYTVOŘENÍ SOUBORŮ

- **mc**edit soubor.txt
- **vi** soubor.txt (**vim** soubor.txt)
- **touch** soubor.txt
- **cat** > soubor.txt (ukončení **Ctrl+D**)

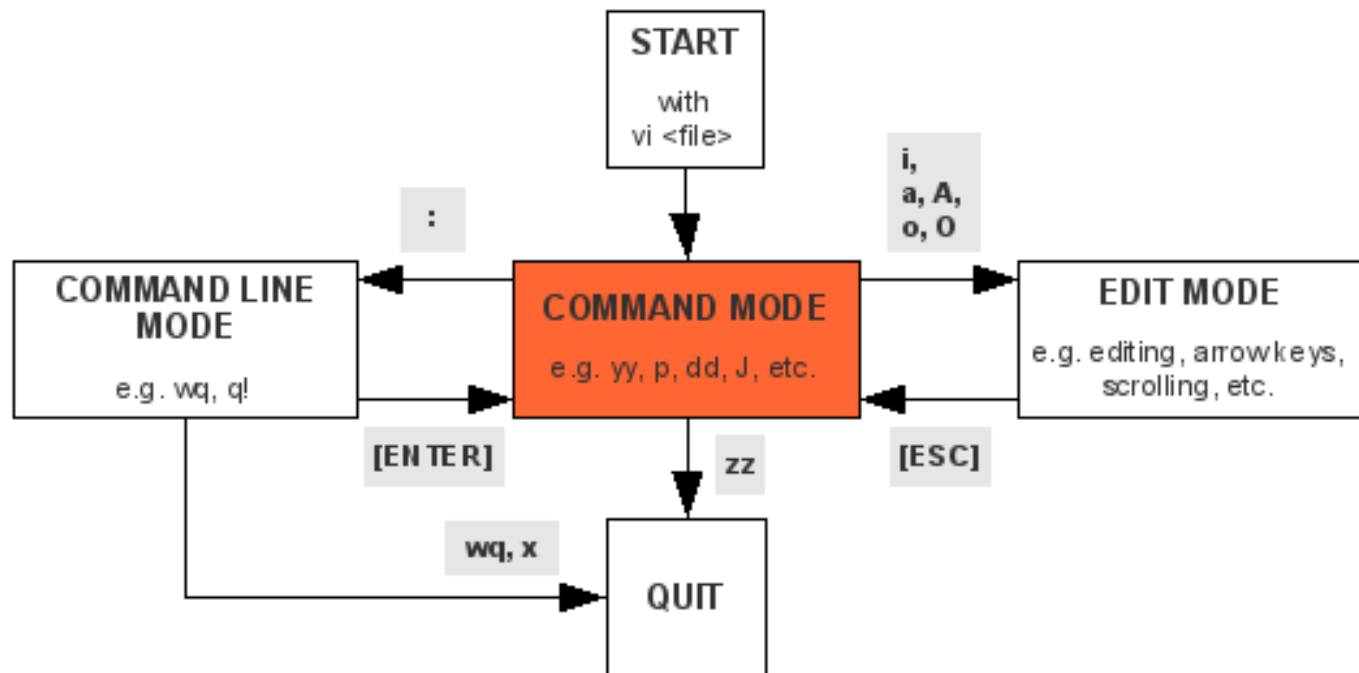
EDITOR VI/VIM/GVIM

vim soubor.txt

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

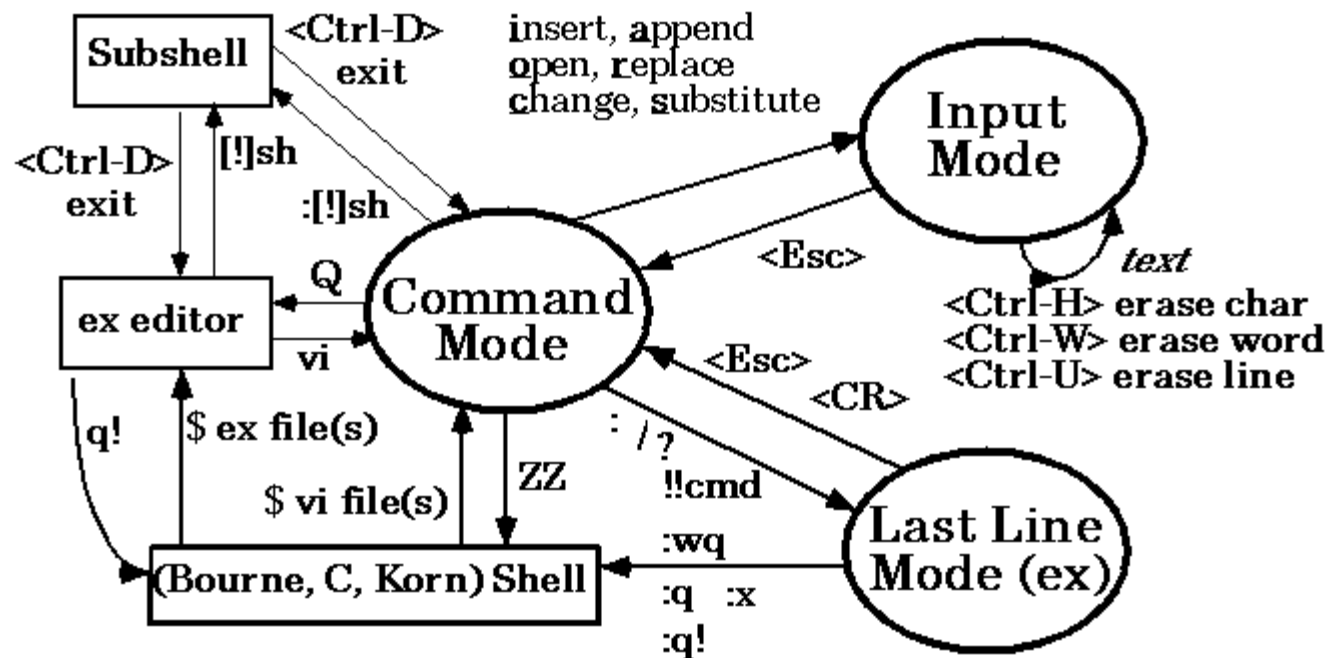
každý musí umět základní
editaci souboru pomocí vi
(vim)

EDITOR VI



EDITOR VI

Vi and Ex modes and States



Zdroj: <http://www.halcyon.com/arkay/Orientation.html>

VI — COPY, CUT, PASTE

- kurzor dáme na začátek bloku
- **v**
- kurzor dáme na konec bloku
- **d** (delete) - cut
- nebo **y** (yank) - copy
- kurzor na požadovanou pozici
- **p** paste

SHRnutí PŘESMĚROVÁNÍ

akce	popis
	Roura – výstup jednoho procesu na vstup dalšího
>	Přesměrování výstupu do souboru
>>	Přesměrování výstupu na konec souboru
2>	Chybový výstup
2>>	Chybový výstup na konec souboru
> vse.txt 2>&l	Oba výstupy do stejného souboru
>> vse.txt 2>> vse.txt	Oba výstupy do stejného souboru
<	Přesměrování vstupu

FILE DESKRIPTORY

- 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`

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`)

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 &
```

// vyzkoušejte i mc &

```
[1] 15260
```

```
eryx4>
```

```
[1] + Suspended (tty output)    top
```

PROCESSY 2.

eryx4> `ps x`

PID	TTY	STAT	TIME	COMMAND
15055	pts/6	S	0:00	-tcsh
15260	pts/6	T	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
(apt install psmisc)

ČÍSLOVÁNÍ ŘÁDKŮ V SOUBORU

- `cat /etc/passwd | nl | head`
 - vypíše prvních 10 očíslovaných řádků
- `cat /etc/passwd | nl | head -n 15`
 - vypíše prvních 15 očíslovaných řádků
- `cat /etc/passwd | nl | tail`
 - vypíše posledních 10 řádků
- `cat /etc/passwd | nl | tail -n 3`
- `cat /etc/passwd | nl | tail -n +3 | more`

lze i
head -15

poslední 3
řádky

od 3.
řádky do
konce

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)

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 — UŽIVATELE

- who

```
pesicka pts/1 Oct 5 23:43 (84.242.95.197)
student6 tty0 Oct 6 00:21 (hyperochus.zcu.cz)
fhoudek pts/3 Oct 6 00:09 (koleje-zcu.souepl.cz)
maskova pts/6 Oct 5 20:25 (b1.sab.plz.sloane.cz)
student6 tty1 Sep 27 14:32 (hyperochus.zcu.cz)
```

- chceme seznam uživatelů, abecedně seřazený a bez duplikací

PŘÍKLAD 1 — UŽIVATELE

- `who | cut -c1-8`

pesicka

student6

fhoudek

maskova

student6

- **vybere jen znaky 1 az 8**

PŘÍKLAD 1 — UŽIVATELE

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

fhoudek

maskova

pesicka

student6

student6

- **setřídí podle abecedy**

PŘÍKLAD 1 — UŽIVATELÉ

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

fhoudek

maskova

pesicka

student6

- **vybere jen znaky 1 az 8**
- **seřídí podle abecedy**
- **odstraní duplicity seřazených řádků**

PŘÍKLAD 1 — PRVNÍ SKRIPT

1. Editace
2. Přidání práva spuštění
3. Vlastní spuštění

PŘÍKLAD 1 — PRVNÍ SKRIPT

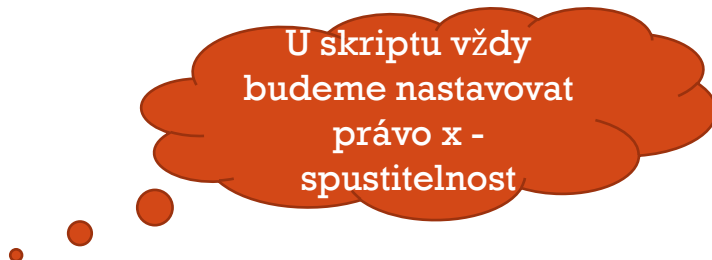
1. `vim` `uzivatele.sh`

```
#!/bin/bash  
who | cut -c1-8 | sort | uniq
```

2. `chmod` `u+x` `uzivatele.sh`

Důležité: skriptu nastavíme právo `x` !

3. `./uzivatele.sh`



U skriptu vždy
budeme nastavovat
právo `x` -
spustitelnost

PŘÍKLAD 2 — 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 3 — 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
```

NAHRAZOVÁNÍ SLOV

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

vypíše:

mala velka pismena

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

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

KOMPRESSE 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

KOMPRESSE 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`

NÁVRATOVÁ HODNOTA PŘÍKAZU

- proces po skončení předává návratovou hodnotu (exit status)
- konvence
 - 0 – úspěšné ukončení příkazu
 - jiné – neúspěch
 - 128+n - ukončení signálem (n je číslo signálu)
- echo \$?
 - Návratový kód posledního příkazu

Při neúspěchu nás zajímá důvod (různé návratové kódy),
při úspěchu stačí status OK.

VYZKOUŠEJTE

sleep 10

Stisknu Ctrl+C

echo \$?

130 při Ctrl+C

(pokud používáme bash, u jiného shellu i jiné hodnoty)

0 když doběhne do konce

SYSTÉMOVÉ PROMĚNNÉ

Výpis proměnných:

- set
- env
- printenv

```
TERM=xterm
UID=1084
USER=pesicka
VENDOR=unknown
```

Proměnná je vidět v aktuálním shellu.

Dostupnost i pro další programy:

`export MYVAR`

EXPORT PROMĚNNÉ

- Příkaz `export MYVAR` – naši proměnnou uvidí i další
- Vytvořte skript `p2.sh`
`#!/bin/bash`
`echo MYVAR ma hodnotu:`
`echo $MYVAR`
- Nyní nastavíme `MYVAR=7` a pustíme náš skript
- Dále provedeme `export MYVAR` a znovu pustíme náš skript

SKRIPT ADRESAR.SH

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

```
echo Jsem v adresari
```

```
pwd
```

```
echo Zmenim na root
```

```
cd /
```

```
pwd
```

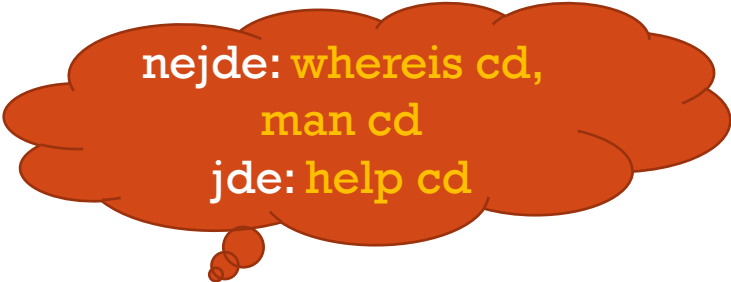
POTOMEK PROCESU

- potomek procesu zdědí **kopii prostředí** svého rodiče (proměnné, akt. adresář)
- potomek může prostředí měnit, ale změny se **nedotknou** rodiče – mění **kopii** původních dat

Př.

1. Vypíšeme aktuální adresář (pwd)
2. Spustíme script adresar.sh
3. Po skončení skriptu je akt. adresář nezměněn

PŘÍKAZY



nejde: **whereis cd,**
man cd
jde: **help cd**

- **vestavěné příkazy shellu** (**cd, set**)
 - nápověda příkazem **help**
- **externí příkazy** (**ls, cp, mv**)
 - **spustitelné** soubory (v **/bin, /sbin, /usr/bin,...**)
 - shell je spustí jako svého potomka
 - nápověda příkazem **man**
- **příkazový soubor** (shell script)
 - očekává textový soubor obsahující příkazy shellu
 - shell spustí svojí kopii a ta provede příkazy

VESTAVĚNÉ X EXTERNÍ PŘÍKAZY

- type who
- type cd

```
eryx3> type who
who is /usr/bin/who
eryx3>
eryx3>
eryx3> type cd
cd is a shell builtin
```

Příkazem type
získáme nápovědu
k vestavěným
příkazům

PŘÍKLAD – ZDRAVÍČÍ SKRIPT

#!/bin/bash

neodřádkuje

echo -n "Zadej sve jmeno: "

read PREZDIVKA

echo Zdravim Te, \$PREZDIVKA

echo Prihlaseny jako \$USER

systémová proměnná

- Vyzkoušejte a odpovězte na otázky na dalším slidu

OTÁZKY K PŘEDCHOZÍMU SKRIPTU

Po spuštění skriptu zadejte příkaz **set**.
Uvidíte proměnnou PREZDIVKA? Zdůvodněte!

Jaký je rozdíl v provedení následujících příkazů?

1. `zdrav_skript`
2. `zdrav_skript Lada` (parametr)
3. `zdrav_skript < vstup.txt` (přesměrování vstupu)

PODMÍNĚNÉ PŘÍKAZY - IF

```
if seznam-prikazu  
then seznam-prikazu  
[ elif seznam-prikazu  
  then seznam-prikazu ] ...  
[ else seznam-prikazu ]  
fi
```

příkazy vždy na novou
řádku

Pokud na stejnou,
oddělovat středníkem

- vykoná se seznam příkazů za if, pokud návratová hodnota 0, provedou se příkazy za then
- if seznam-prikazu ; then seznam-prikazu ; fi

PŘÍKLAD – IF1.SH

```
#!/bin/bash
# skript testuje, co je parametr
if test -d "$1"
then
    echo "$1 je adresar"
elif test -f "$1"
then
    echo "$1 je obycejny soubor"
else
    echo "$1 neni adresar ani obycejny soubor"
fi
```

test -d file
test na adresář

test -f file
test na obyčejný
soubor

Lze zapsat:
[-f file]
důležité jsou mezery
kolem závorek

PŘÍKLAD IF2.SH – TEST PINGU

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

```
echo "Zadej jmeno stroje: "  
read STROJ  
if ping -q -c 3 $STROJ > /dev/null  
then  
    echo "Stroj $STROJ pinguje"  
else  
    echo "Stroj $STROJ nepinguje"  
fi
```

Praktický
příklad,

Využívá
návratový kód
příkazu ping

PODMÍNĚNÝ PŘÍKAZ - CASE

case slovo in

vzor) seznam-prikazu ;;

vzor) seznam-prikazu ;;

...

esac

- srovnává slovo se vzorem
- pokud souhlasí, vykoná seznam příkazů a skončí

PŘÍKLAD – CASE1.SH

CASE, PÍPNUTÍ, SYSTÉMOVÁ PROMĚNNÁ

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

```
# skript zpracuje první parametr
```

```
case $1 in
```

```
-h | -help) echo " Napoveda: spust s jedním parametrem"
```

```
    echo " -c, -d, -p nebo -b"
```

```
    ;;
```

```
-c)    echo "Kalendar: " ; cal ;;
```

```
-d)    echo " Dnes je:" ; date ;;
```

```
-p)    echo -e " Ted pipnu... \a " ;;
```

```
-b)    echo " $USER je borec " ;;
```

```
*)    echo " Neznámá volba, zkus -h"
```

```
    ;;
```

```
esac
```

PŘÍKAZ CYKLU - FOR

for proměnná in seznam-slov

do

seznam-příkazů

done

- nejdříve expandován seznam slov
- oddělena mezerou

PŘÍKLAD – FOR1.SH

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

```
for den in patek sobota nedele
```

```
do
```

```
    echo Oblíbený den je $den
```

```
done
```

PŘ. FOR2.SH

JMÉNA PODADRESÁŘŮ V AKTUÁLNÍM ADRESÁŘI

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

```
for X in *  
do  
    if test -d "$X"  
    then  
        echo "Podadresar: $X"  
    fi  
done
```

Proč je \$X v uvozovkách?
Zkuste si, zda vám skript
bude fungovat i na
adresář
mkdir "red five"

FOR S TŘEMI VÝRAZY (FOR6.SH)

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

```
for (( c=1; c<=5; c++ ))
```

```
do
```

```
    echo "Welcome $c times"
```

```
done
```

FOR – NEKONEČNÁ SMYČKA (FOR7.SH)

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

```
for (( ;; ))
```

```
do
```

```
    date
```

```
    echo "stiskni CTRL+C"
```

```
    sleep 2
```

```
done
```