Operační systémy - informace

Náplň

- Úvod do principů operačních systémů, prezentovaný na OS Unix (základní informace a principy, příkazy)
- Praktické procvičování (vzdáleně na serverech FIT)
- Odborný týden na FIT
- Administrace unixových systémů (teorie)
- Základy sítí
- Příprava na maturitu

Literatura

- Cokoliv, kde se v názvu vyskytuje „Základy Unixu“, „Úvod do Linuxu“ apod.
- Např.: Operační systém Linux, Vilém Vychodil, Computer Press, Brno, 2003
  http://vychodil.inf.upol.cz/errata/download/linux-pcu-kap5.pdf

Klasifikace

- Aktivita v hodinách (+,-)
- Malé písemky
- Čtvrtletní písemky
- Zkoušení (pouze v mimořádných situacích)

Kontakt

Zdeněk Muzikář: zdenek.muzikar@gyarab.cz nebo zdenek.muzikar@fit.cvut.cz

O předmětu

Proč učit operační systémy na Unixu?

- Návrh je tak čistý, elegantní a převratný, že posloužil jako vzor všem ostatním současným operačním systémům.
- Unix je od počátku spjatý s univerzitním prostředím a velkou část vývoje provedly právě univerzity s ohledem na své potřeby (mnoho uživatelů, robustnost, bezpečnost…).
- Jedná se o otevřený systém, pro řadu variant Unixu jsou dostupné kompletní zdrojové kódy a lze tak neomezeně pronikat do všech zákoutí a objevovat jejich krásu ☺

Proč budeme pracovat vzdáleně na Solarisu?

- Solaris je jedna z variant Unixu, komerčně sice zaostává např. za HP-UX nebo AIX, ale ve vývoji je nejdále. Faktem nicméně je, že na úrovni tohoto předmětu rozdíly od Linuxu prakticky nepoznáte.
- Budete na tom všichni stejně a nevzniknou problémy s různými distribucemi Linuxu.
- Můžete se vzdáleně přihlašovat a vše procvičovat 24 hodin denně 365 dní v roce.

Vlastnosti OS UNIX

- Víceúlohový (multitasking, time-sharing)
- Víceuživatelský (multiuser)
- Přenositelný
- Interaktivní i dávkový přístup (shell)
- Přesměrování a řetězení vstupu a výstupu
- Hierarchický systém souborů
- Podpora práce v síti
- Grafické prostředí

Architektura UNIXu

- **hardware**
- **kernel** (jádro)

  funkce volání jádra, ovladače, démoni, virtuální paměť (swap),
  plánování procesů, systémy souborů,…

- **shell**

  Bourne shell, Korn shell, bash, Z shell, Posix shell - Bourne shell family *
  C shell, tc shell - C shell family
  restricted shell, dt shell, …

  2 funkce:
  - uživatelské rozhraní, interpret příkazů
  - interpret programovacího jazyka

*/ „rodiny“ se od sebe odlišují syntaxí příkazového jazyka (a la Pascal x C)

Komunikace s uživatelem

Grafické prostředí (Desktop Environment) GUI

- KDE, CDE, JDE (Gnome),…
- nastavení prostředí
  - chování a vlastnosti, zapamatování (jak?)
  - current session, home session

Příkazová řádka (shell) CLI

- jednotná struktura: příkaz, přepínače, argumenty
- oddělovače částí příkazu: mezera, tabulátor
- oddělovače příkazů: nová řádka, středník
- rozlišování velikosti písmen (case sensitive)

Přihlášení

- Lokálně do grafického prostředí
  výběr prostředí, nastavení, session
- Lokálně do příkazové řádky
  program login
- Vzdálené přihlášení – klient, server
  ssh, telnet, rlogin

- Přihlašovací shell (login shell) bude později
- Inicializační soubory
- Domovský adresář
- Odhlášení (ukončení přihlašovacího shellu)

Vzdálené přihlášení - I

- Protokol **ssh** (secure shell) - kryptovaná komunikace
- Model „client“ (využívá službu) – „server“ (poskytuje službu)
- Na vzdáleném počítači „daemon“ - **server**
  - Běží neustále od startu služby (systému)
  - Obsluhuje příchozí požadavky na bezpečné spojení
  - Jméno obvykle **sshd** a poslouchá na TCP portu 22
  - Server se identifikuje otiskem (finger print) ~/.ssh/known_hosts
- Na lokálním počítači uživatel spouští program - **client**
  - Obvykle programy **ssh** (Unix) nebo **putty** (MS Windows)
  - Program naváže šifrované spojení se serverem bude později
  - Klient (uživatel) se autentikuje jménem a heslem, případně klíčem
  - Po úspěšné autentikaci se spustí login shell
  - Ukončení spojení – skončení login shellu, delší výpadek sítě

Vzdálené přihlášení - II

Navázání spojení: **`ssh username@hostname`**

Příklady:

```
ssh honza@fray1.fit.cvut.cz
ssh -l honza fray1.fit.cvut.cz
```

program putty v MS Windows

Ukončení spojení (shellu - záleží o jaký se jedná)

```
exit
logout
bye
^D  (ctrl D)
kill -1 $$     # bude vysvetleno pozdeji
```

Terminálové okno a příkazová řádka (CLI)

- **výzva** — prompt
  zobrazována shellem, lze nastavit proměnnou PS1
- **jméno příkazu** — command name
  určuje, který příkaz se vykoná (co)
- **přepínače** (volby, modifikátory) — options
  ovlivňují vykonání příkazu (jak)
- **argumenty** (parametry) — arguments
  specifikují data ke zpracování (s čím)

přepínače a/nebo argumenty mohou chybět

Příklady:

```
$ date                          # # je komentar
# date 1040                     # smi pouze root
$ date; sleep 2; date           # ; je oddelovac
$ ls -lR /usr/bin /tmp          # za - jsou prepinace
```

Příklady příkazů

```
$ banner Ahoj       # v Linuxu není, lze nainstalovat
$ echo Dobry den
$ cal ; cal 2010 ; cal 12 2010
$ who ; who –H
$ sleep 2 ; clear ; sleep 3; banner Baf
$ last -20
$ cat /etc/resolv.conf     # bude pozdeji
$ ls ; ls –l ; ls –la
$ hostname
$ whoami
$ who am i
$ prstat                   # ukonceni q nebo <ctrl> C
$ ps –f                    # procesy aktivniho okna
$ history
```

Vzdálené spuštění příkazu a bezpečný přenos souborů (pomocí protokolu ssh)

Spuštění příkazu na vzdáleném počítači:

`ssh uname@hostname` command

Pravidla:

- Příkaz **command** se spustí na počítači **hostname** pod identitou **uname**
- Je vyžadováno heslo uživatele **uname** nebo je třeba mít nastaveny klíče
- Není-li uvedeno jméno uživatele, převezme se jméno stávajícího
- Není-li uveden příkaz, provede se “login“ (vzdálené přihlášení)

Příklad:

```
ssh honza@fray1.fit.cvut.cz hostname
```

Bezpečný přenos souborů

- Vzdálené kopírování souborů - alespoň jeden soubor je mimo systém, na kterém uživatel pracuje
- Využívá se protokol ssh
- Příkaz **scp** (Unix) nebo **winscp** (MS Windows)
- Schéma příkazu: **scp** ***zdroj cíl***
- Struktura vzdáleného zdroje (cíle):

  `username@hostname:pathname/filename`

Pravidla:

- Stejná, jako při vzdáleném spouštění příkazu – viz dříve
- Je-li **pathname** relativní, začíná ve “vzdáleném home“ uživatele
- Je-li cílem adresář, zůstává původní jméno

Příklady:

```
scp dopis.txt soused@192.168.1.10:Dokumenty/od_Honzy.txt
scp fray1.fit.cvut.cz:pisemka.txt .
```

Základní termíny

- běžný uživatel a administrátor (root – může vše)
- uživatelské jméno a heslo
  identita vnější (uživatelské jméno) a vnitřní (UID)

  `id -a`
- přihlášení/odhlášení
- domovský adresář, pracovní (aktuální) adresář
- soubor (podrobnosti později)
  - absolutní cesta – vzhledem vrcholu adresářového stromu
    začíná / (adresář root – neplést s administrátorem root)
  - relativní cesta - vzhledem k pracovnímu adresáři
    nezačíná /
- adresář (je to také soubor – podrobnosti později)

Prompt

- pro „Bourne shell family“ definován v proměnné **PS1**
  pro csh v proměnné **prompt**
  (bude vysvětleno později)

- implicitní nastavení různé pro běžného uživatele (**$**) a roota (#)

- lze snadno měnit,
  uživatel si jej může v konfiguračním souboru nastavit,
  u vyšších shellů (ksh, bash) může mít dynamické vlastnosti

Jméno příkazu

- příkaz může být buď **vnější** (soubor) nebo **vnitřní** (zabudovaný v shellu), podrobný popis je v manuálových stránkách

- vnější bez cesty – vyhledání řízeno proměnnou **PATH** a soubor musí být spustitelný bude později

  Př.: **`banner ahoj`**
  `which which`

- vnější včetně cesty (není třeba nastavit **PATH**)

  Př.: **`/usr/bin/banner nazdar`**
  `../program1`
  `./program2`

- jméno nemusí být jedinečné! Pak rozhoduje pořadí v **PATH**!

- příkaz vyhledává shell (podle proměnné **PATH**)
  nenalezne-li: `command not found`
  nemá-li uživatel oprávnění: `Permission denied`

Přepínače

- mohou chybět
- většinou začínají znakem **–** občas **+**
- obsahují právě jeden znak (nejedná-li se o „dlouhé“)
- „dlouhé“ přepínače (Linux):
  Př.: **`--reverse`**
- lze je psát zvlášť nebo dohromady
  Př.: **`ls -l -a ; ls -la ; ls -al`**
- na pořadí někdy záleží, někdy ne
  Př.: **`sort +2 -3 +4 -5 soubor`**
  `sort +4 -5 +2 -3 soubor`
- mohou mít jeden argument
  Př.: **`tar -cvf /var/tmp/archiv .`**

```
# soubor archiv je argumentem přepínače f
# adresář . je argumentem příkazu tar
```

Argumenty

- počet je definován příkazem, někdy proměnný, mohou i chybět
- většinou soubory nebo textové řetězce

Příklady:

```
clear ; date ; hostname
cat /etc/passwd ; banner /etc/passwd
echo Jak se mas
cal ; cal 2010 ; cal 10 2010
pwd ; id –a
who ; who -H
sleep 10
sleep 100000
^C
```

násilné přerušení příkazu

Manuálové stránky

- exaktní popis příkazu
  syntaxe: man <jméno příkazu>

  Př.: **man sort**
  man man

- vnitřní příkazy: **?** *mezera* *enter* **b** **/** **n** **q** .....

- sekce: **man exec**
  man -s 2 exec

- další přepínače
  man -a exec
  man -k assembler
  man -f /etc/passwd

Příkaz `ls`

- **vypisuje informace o souborech**

- **2 režimy** a/ argument soubor – vypíše informace o souboru
  b/ argument adresář – vypíše informace o souborech v něm

- **implicitní chování** (zavolán bez přepínačů, bez argumentů)
  vypíše seznam souborů v aktuálním adresáři, ignoruje skryté soubory

- **některé přepínače** (lze kombinovat)
  - **`-l`** **l**ong listing – vypíše podrobnosti o souborech
  - **`-a`** **a**ll – vypíše všechny soubory, tedy i skryté
  - **`-d`** **d**irectory – je-li argumentem adresář, vypíše informace o něm, nikoli o souborech v něm (převede režim b/ na a/)
  - **`-R`** **R**ecursive – vypíše zadaný adresář a rekurzivně všechny podřízené

- **Příklady**

```
$ ls -la /etc/security    # soubory serazeny abecedne
$ ls -ld                  # totéž jako:  ls -d -l .
$ ls -lRa .
$ ls -l .. /etc/cron.d    # vypisuje adresare postupne
```

Příkazy `cat` a `file`

cat **vypisuje obsah (textového!) souboru**
ve skutečnosti pouze filtr, který kopíruje vstup na výstup – bude později

některé přepínače
- `-n` čísluje řádky
- `-b` čísluje jen neprázdné řádky

file **odhadne (heuristika!) typ souboru**

Příklady

```
$ cat /etc/profile
$ cd /etc; cat -n profile; cat -b profile
$ file .
$ file /etc/profile /etc/.login
$ file /bin/file
```

Příkazy `od` (octal dump) `a strings`

od **vypisuje obsah souboru po bytech** – i binární soubory

některé přepínače definující formát:

`-odxc` oktalový, dekadický, hexadecimální, znakový

strings **vypisuje pouze tištitelné části souboru**

Příklady

```
$ echo ahoj >hoj.txt ; ls -l hoj.txt
$ cat hoj.txt ; strings hoj.txt ; od -xc hoj.txt
$ od -x /bin/strings
$ strings /bin/strings
$ cat /bin/strings              # radeji nezkouset ☺
```

Otázky

Jak to, že příkaz **ls –l hoj.txt** vypisuje délku v bytech 5 ?

Jak se liší formát textového souboru v unixových systémech a MS Windows ?

Zpracování příkazové řádky

- Shell přečte příkazovou řádku
- V několika krocích ji zpracuje a provede různé transformace (viz dále)

Vnitřní příkaz provede shell sám

při kolizi jmen má přednost před vnějším příkazem

Vnější příkaz (spustitelný soubor) zavolá

- binární program - vzniká překladem ze zdrojového programu
- skript (dávka) – textový soubor s příkazy

obojí se provádí v jiném procesu!

Neexistuje-li příkaz, ohlásí shell chybu

„`command not found`“

Aliasy

- Příkazové zkratky (ne Bourne shell)
- Definuje uživatel
- Tabulka (2 sloupce) uvnitř shellu
- Textová substituce v shellu
- Nedědí se
  trvalá definice: konfigurační soubor shellu, např. : **`.bashrc`**

Příklady:

```
alias c=clear
alias ll=´ls -l´
alias DATE=´c; date; sleep 5; date´
alias
unalias DATE
```

Příkaz - filtr

- Většina příkazů má strukturu filtru
  - nezabývají se fyzickou reprezentací vstupu a výstupu
  - řeší shell
- Standardní vstup, výstup, výstup chyb
  implicitní přiřazení (klávesnice, monitor)
- Deskriptory – čísla „kanálů“
- Přesměrování vstupu a výstupu (>, >>, <, <<)
  - příkaz **m>soubor** (m,n – deskriptory)

    příklad:
    ```
    ls -lR /etc 2>/dev/null    # chyby zmizí
    echo Dobry den >dopis.txt  # vznikne soubor
    date 1>>dopis.txt          # pridat na konec
    ```
  - příkaz **m>&n**

    příklad:
    ```
    ls -lR /etc 1>vse.txt 2>&1 # chyby+data dohromady
    ```
- Výhody koncepce: zařízení = soubor

  příklad:
  ```
  banner kuk >/dev/pts/99 # vystup v jinem okne
  ```

Užitečné příkazy (filtry)

grep <*vzor*> [soubor]

výpis řádků obsahující zadaný vzor (podrobně bude později)

některé přepínače:

- **-v** invertuje smysl (výpis řádků **ne**obsahující vzor)
- **-i** nerozlišuje velká a malá písmena

sort [soubor]

třídění (podrobně bude později)

implicitně alfanumericky od začátku řádky

některé přepínače:

- **-n** numerické třídění
- **-t<*znak*>** definice oddělovače polí (implicitně mezery a tabulator)
- **-k<*n*>,<*n*>** třídění podle <*n*> tého pole

tee <*soubor*>

posílá vstup beze změny na výstup a zároveň do *souboru*

Kolony příkazů

- Standardní výstup (1) příkazu se propojí se standardním vstupem (0) následujícího.
- Celkový výstup je výstup posledního příkazu.
- Návratový kód kolony je návratový kód posledního příkazu (bude později).
- Všechny příkazy se spustí paralelně!
- Příkaz „neví“, že pracuje v koloně

Příklady:

```
ypcat passwd | grep novak | cat -n >novakove.txt
who | grep ar | tee uzivatele | wc -l
ps -f | sort
```

Další příkazy (filtry)

more [**–*n***]
výpis textu po **n** řádkách (implicitně n=<počet řádek okna>)

```
ls -lR /etc | more
```

head –*n*
výpis prvních **n** řádek souboru/std.vstupu

tail –*n* ; **tail +*n***
výpis posledních **n** (-n) řádek nebo od **n-té** řádky do konce (+n)
je-li po čísle ***n*** písmeno **b**, jedná se o bloky (512byte), pokud **c** jde o znaky

```
tail -3c /usr/dict/words
tail -1b /usr/dict/words | wc
ls –l | tail +2
```

cut –c<*seznam*>
cut –d<*znak*> -f<*seznam*>
„vykrojení“ znaků nebo polí podle seznamu

```
ls –l | cut –c1-10,54-
ypcat passwd | cut –d: -f1 | head -100 | more
```

Znak „~“

Znak **~** (tilda) expanduje **ksh** a **bash** takto:

~	- domovský adresář uživatele
~usernamex	- domovský adresář uživatele **usernamex**
~-	- předchozí pracovní adresář

Příklady:

```
cat ~/.bashrc
echo ~root
cd ~root
cd ~-
cd ~-
```

Proměnné I

- Definovány na úrovni shellu
- Označeny identifikátorem (case sensitive!)
- Typ je „řetězec“ (ksh a bash umí i celá čísla a pole)
- Definice: `PI=3.14  # ne mezery!`
- Použití:
```
echo Ludolfovo cislo je $PI
echo Presneji: PI=${PI}1592653
```
- Výpis všech: `set`
- Zrušení: `unset PI`
- Některé proměnné (PATH, PS1, HOME, MAIL,…) jsou předdefinovány automaticky a ovlivňují chování shellu nebo jiných programů

Proměnné II

- Proměnnou lze „exportovat“
- Tím se stává součástí prostředí (environment)
- Kopíruje se (dědí) do potomků – ovlivňuje je (např. **`PAGER` ↴ `man`**)
- Z potomka nelze žádným způsobem změnit prostředí rodiče
- Definice: **`export PI`**
- Výpis: **`env ; export`**

- ksh a bash mají celočíselné a „read only“ proměnné
- Příklady:

```
typeset -i NUMBER
typeset -i8 OCTAL=9     # syntaxe ksh
NUMBER=100; CISLO=sto; echo $OCTAL
typeset -r NUMBER CISLO JMENO=Jirka MESTO=Praha
OCTAL=16#FF; echo $OCTAL   # ksh
```

Substituce příkazů

- Řetězec ozavřený mezi ` ` provede shell jako příkaz a vše nahradí standardním výstupem tohoto příkazu
- V shellech **ksh** a **bash** je možno použít i **$( )**

Příklady:

```
$ echo dnes je `date`
$ HOST=$(hostname)
$ echo pracuji na pocitaci $HOST na Solarisu \
verze `uname -r`
$ CITAC=10
$ CITAC=`expr $CITAC + 1`
$ echo Na pocitaci $HOST prave pracuje \
`who|cut -c1-8|sort|uniq|wc -l` uzivatelu
```

Expanze metaznaků

Metaznak – zástupný znak (v shellu) při popisu jména souboru

*****	libovolný (tedy i 0) počet libovolných znaků, kromě **.** v 1. pozici
?	právě jeden libovolný znak, kromě **.** v 1. pozici
[ ]	právě jeden znak z množiny znaků definované v závorkách

lze definovat:

- výčtem **[abcmn]**
- intervalem **[d-m]**
- kombinací obojího **[abc-n:(z]**

[!]	právě jeden libovolný znak, kromě znaků v závorkách a **.** v 1. pozici

Příklady:

```
$ cd /usr/bin ; echo l*n
$ echo [ad]*[o-z]?? ; echo *[!a-z\-]*
$ echo .* ; ls .*          # proč se ls chová "divně"
`
```

Význam znaků ’ ” \

\ chrání před shellem právě 1 následující znak

’….’ chrání před shellem řetězec znaků

”….” chrání před shellem řetězec znaků, kromě **$ ` **

Příklady:

```
$ echo  $HOME \$HOME  \\  \”
$ echo  ” ’ ”   ’ ” ’
$ echo   date .* `date` $HOME \$HOME
$ echo  ’date .* `date` $HOME \$HOME’
$ echo  ”date .* `date` $HOME \$HOME”
```

Analýza příkazové řádky shellem

1. Přečtení příkazové řádky a rozdělení na části ($IFS)
2. Zpracování klíčových slov (if, then, for…)
3. Zpracování aliasů (ne sh)
4. Provedení vnitřních příkazů (alias, exit…)
5. Provedení funkcí
6. Expanze znaku ~ “tilda” (ne sh)
7. Substituce příkazů - ``, $( )
8. Substituce proměnných ($PATH, $TZ)
9. Substituce aritmetických výrazů (ne sh) (( ))
10. Expanze metaznaků – jmen souborů (a*[0-9]?)
11. Vyhledání příkazu
12. Provedení příkazu

Implementace souboru na úrovni metadat

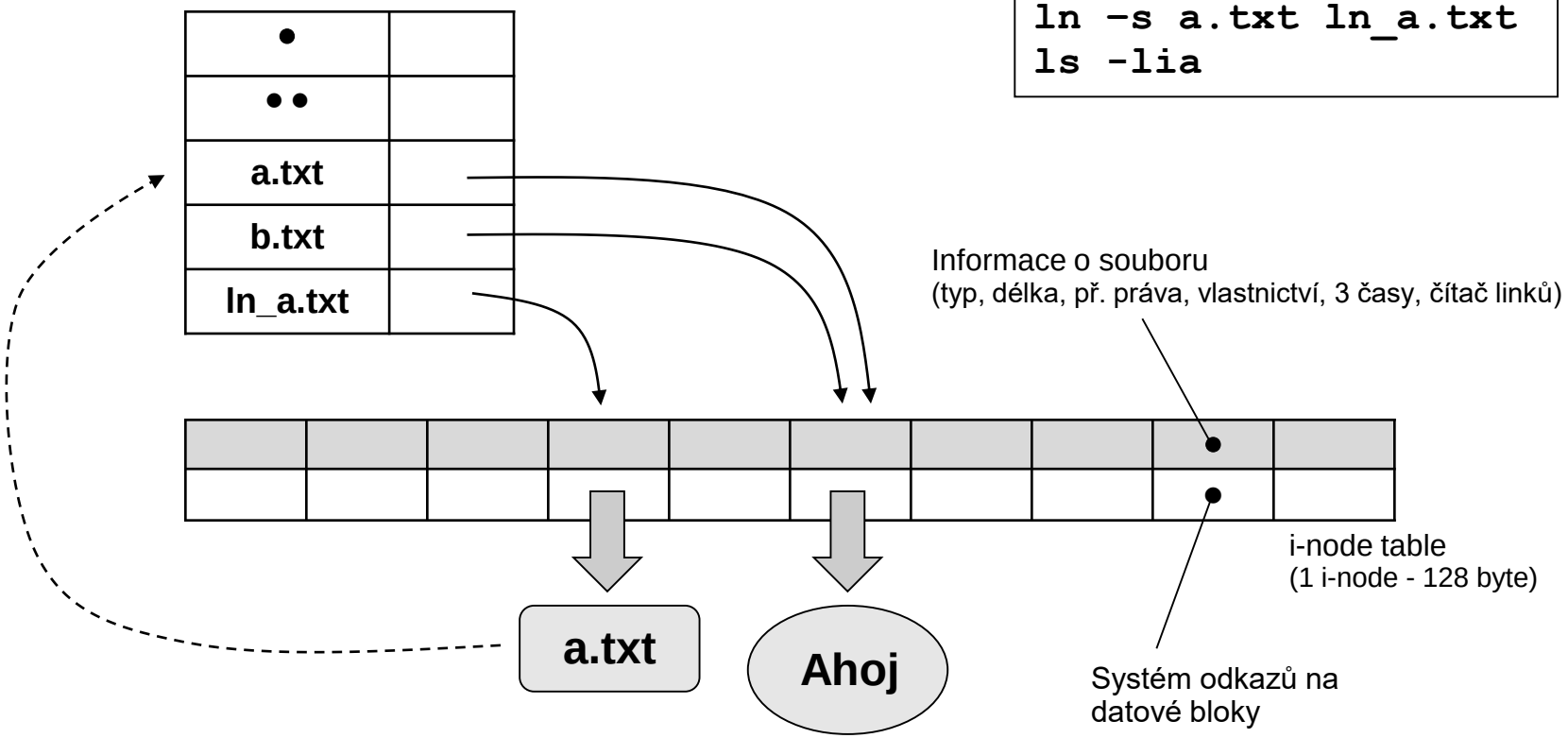
- Adresář (tabulka o 2 sloupcích) – jméno a odkaz na i-node
  (rozdíl mezi kopírováním (cp) a přemístěním (mv) souboru)
- i-node (informační uzel) – tabulka
- i-node obsahuje informace o souboru a odkazy na datové bloky
- Pevné linky

  Omezení pevných linků
  - **NE** mezi různými systémy souborů
  - **NE** na adresáře
- Symbolické linky
- Typy souborů

  obyčejný soubor (-), adresář (d), symbolický link (l), speciální soubor (c nebo b), roura (p)

Implementace souboru (na úrovni metadat)

```
mkdir dopisy
cd dopisy
echo Ahoj > a.txt
ln a.txt b.txt
ln –s a.txt ln_a.txt
ls -lia
```



Pevný a Symbolický link

Pevný link

- z adresáře(ů) vede více odkazů na stejný i-node (více jmen pro tatáž data) - viz **a.txt** a **b.txt** výše
- stejná úroveň - není rozdíl mezi „původním“ a „novým“ jménem
- počet se uchovává v čítači linků, hlídá systém
- nelze použít pro adresáře
- nemůže vést přes hranici fyzického systému souborů

Symbolický link

- v datech je jméno jiného souboru – textový odkaz – viz **ln_a.txt**
- neplatí omezení pro pevný link
- systém nekontroluje konzistenci
- lze vytvořit i na neexistující soubor (zjistí se až při použití)

Typy souborů

- **Obyčejný soubor** (binární nebo textový)
  v datových blocích obsahuje data
- **Adresář**
  soubor, jehož data tvoří tabulku o 2 sloupcích:
  jméno souboru a odkaz (index) na i-node
- **Symbolický link**
  v datech je textový odkaz na jiný soubor
  (něco jako zástupce v MS Windows)
- **Speciální soubor** (blokové (b) nebo znakové (c) zařízení)
  v i-node odkaz na driver – velké (MA) a malé (MI) číslo zařízení
- **Roura**
  soubor typu FIFO

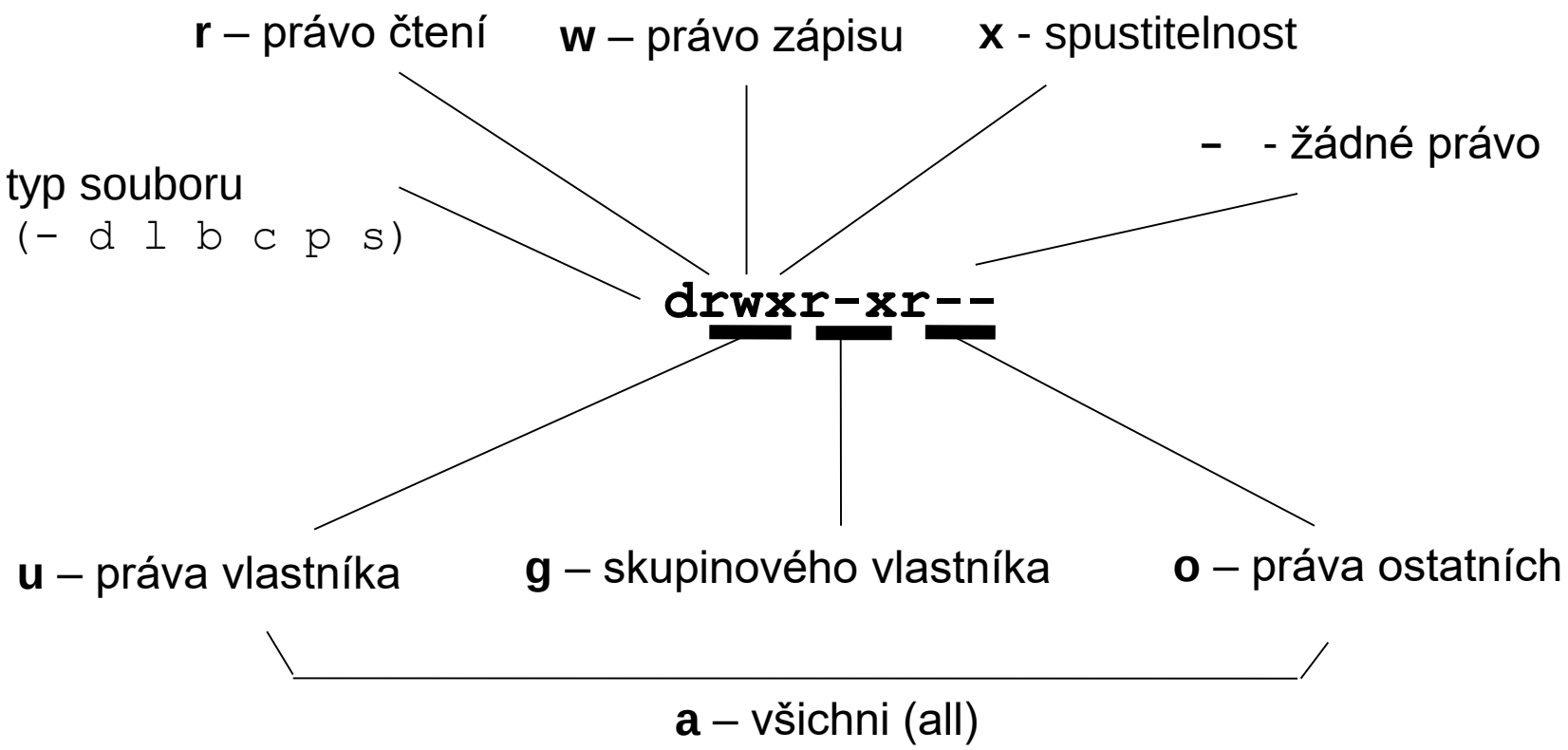
Uživatel a skupiny uživatelů

- Uživatel je definován v **/etc/passwd** nebo jmenné službě (NIS, LDAP…) – atributy: uživatelské jméno, UID….
- Uživatelé se sdružují do skupin
- Skupiny jsou definovány v **/etc/group** nebo jmenné službě
- Uživatel patří do jedné **primární** skupiny (určena v /etc/passwd) a libovolně **sekundárních** skupin (určeny v /etc/group)
- Soubor je vlastněn uživatelem a skupinou
- Nový soubor je vlastněn tím, kdo ho vytvořil a jeho aktuální skupinou

Příkazy:

```
$ groups [username]
$ id -a [username]
$ touch pokus1 ; ls -l
$ newgrp nova_skupina ; id -a
$ touch pokus2 ; ls -l
$ chgrp stara_skupina pokus2 ; ls -l
```

Přístupová práva



Nastavení přístupových práv

Příkaz: **chmod** *výraz* **soubor(y)**

symbolický (relativní) oktalový (absolutní)

Symbolický režim

kategorie práva: **u** (vlastník), **g** (skupina), **o** (ostatní), **a** (všechny kategorie)
operace: **+** (přidat), **–** ubrat, **=** (nastavit)
právo: **r** (čtení), **w** (zápis), **x** (spustitelnost, resp. vstup do adresáře)

platí různé „defaulty“, nepříliš systematické

Oktalový režim

váhy: r = 4, w = 2, x = 1
např.: **`rwxr-xr-- (754), rw-r----- (640)`**

Příklady:

```
$ mkdir dopisy; cd dopisy; echo Ahoj > d.txt
$ chmod u-r+w,g+rwx,o= d.txt ; ls -l  # --w-rwx---
$ cat d.txt                           # nezdari se
$ chmod 600 d.txt ; cat d.txt         # zdari se (rw-------)
$ chmod 177 . ; ls -l                 # nezdari se
```

Procesy I

- Proces je instance spuštěného programu.
- Veškeré aktivity, které v systému probíhají, provádějí procesy.
- Některé procesy běží v systému trvale (tzv. démoni) a zajišťují nejrůznější služby.
- Procesy mají řadu atributů (číslo, rodič, vlastník, stav, priorita…..).
- Vlastníkem procesu je (většinou) uživatel, který proces spustil.
- Přístupová práva jsou vyhodnocována vzhledem k procesům.
- Procesy dynamicky vznikají a zanikají a jsou hierarchicky uspořádány (vztah rodič-potomek).
- Může běžet více procesů, či úloh, paralelně (tzv. multitasking), vzniká problém s prioritou

Příkazy:

```
$ ps ; ps -f      # výpis procesů běžících v aktuálním okně
$ ps -f; ps -l    # výpis více atributů (tzv. full nebo long listing)
$ ps -ef          # výpis všech (every) procesů ve full formátu
$ top             # dynamický výpis
```

Procesy II – signály

- S procesy lze komunikovat (nejčastěji je ukončovat) pomocí signálů
- Signály se posílají procesu příkazem: **kill –č.signálu č.procesu**
- Výpis všech signálů: **kill -l**
- Některé důležité signály: 1 (HUP), 2 (INT), 9 (KILL), 15 (TERM), 24 (STP)
- Některé signály jsou namapované na klávesy: **2-<ctrl>C**, **24-<ctrl>Z**
- Lze zjistit, či změnit, příkazem **stty**

Příkazy:

```
$ kill -9 1234    # násilné ukončení procesu číslo 1234
$ stty -a         # výpis nastavení driveru klávesnice
$ stty intr ^A    # změna vyslání signálu 2 na stisk <ctrl>A
```

Procesy III – úlohy

- Úloha je, zjednodušeně řečeno, uživatelem spuštěný program.
- Je „nad“ procesem – může (ale nemusí) být tvořena více procesy. Např. **sleep** je úloha tvořená 1 procesem, spuštěný skript většinou více.
- Úlohy jsou relativní vůči uživateli, na rozdíl od procesů, které jsou v systému absolutní.
- Lze je spouštět tzv. „na pozadí“, příp. přesouvat „na popředí“ a zpět.
- Běžící úlohu lze signálem č. 24 pozastavit a znovu spustit příkazem **fg** (na popředí – foreground) nebo **bg** (na pozadí – background).

Příkazy:

```
$ sleep 1000 &      # spuštění úlohy na pozadí
$ sleep 2000        # spuštění úlohy na popředí
$ <ctrl>Z           # pozastavení úlohy běžící na popředí
$ jobs              # výpis všech úloh
[1]   Running   sleep 1000 &
[2]+  Stopped   sleep 3000
$ bg %2             # spuštění úlohy 2 na pozadí
$ fg %1             # přehození úlohy 1 na popředí
```

Skripty

- Soubory obsahující příkazy
- Mají charakter programu
- Obsahují „výkonné“ a „řídící“ (podmínky, cykly…) příkazy
- Jako celek se chovají jako filtry (standardní příkazy)
- Jsou prováděny (interpretovány) shellem

Možnosti spuštění:

```
1) ksh muj_skript.sh        # bude proveden shellem ksh
2) csh < muj_skript.sh      # bude proveden shellem csh
3) ./muj_skript.sh          # musí být v aktuálním adresáři a spustitelný
4) muj_skript.sh        # musí být v některém adresáři uvedeném v PATH a spustitelný
```

V případě 3) a 4) je shell definován na 1. řádku, jinak se dědí:

```
#!/usr/bin/bash
echo ”Tento skript provadi nejspis bash, viz:” `ps -f`
echo Pokud ne, byl spusten podle 1\) nebo ”2)”
```

Parametry skriptu (shellu)

- Parametry se do skriptu dostávají přes tzv. poziční parametry (proměnné)
- Uvnitř skriptu jdou nastavovat příkazem set
- V shellech sh, ksh, bash: **`$0, $1,... $9, $#, $*`**
- Pouze v ksh a bash: **`${10}, ${11},....`**

Příklad – napište skript **testik**:

```
#!/bin/bash
 echo ″Jmeno skriptu je:  $0″
 echo ″Pocet parametru skriptu je:  $#″
 echo ″5. parametr je v promenne \$5 a ma hodnotu: $5″
 echo 12. parametr je v promenne ’${12}’ a je: ${12}
 echo ″Vsechny parametry jsou $*″
 echo ″Tento skript $0 provadi proces $$ (viz: `ps –f`)″
```

Zavolejte jej:

```
testik Ted je `date`
ksh < testik ″Ted je″ `date`
```

Příkaz test

Vnitřní příkaz shellu

Testuje 3 kategorie relací:

- Relace řetězců = != (< > … jen bash a ksh)
- Relace čísel -eq -ne -lt -le -gt -ge
- Atributy souborů -d -L -f -r -w -x …….. (viz **man bash** (!) , ale i **man test**)

Příklady:

```
I=3
test $I = 3 ; echo $?                    # bude 0
[ $I = 03 ] ; echo $?                    # bude 1, jiná syntaxe téhož
[ $I -eq 03 ]   ; echo $?                # bude 0
test –d /etc/passwd  ; echo $?           # bude 1
[ -d . ]    ; echo $?                    # bude 0
```

Návratový kód

Návratový kód příkazu

je obsahem proměnné **$?** vždy po provedení příkazu

význam: = 0 … úspěch, příkaz se povedl, OK

≠ 0 … něco špatně, číslo označuje důvod

pravidla:

1 … příkaz pracoval, ale neuspěl

2 … špatné použití (neexistující soubor, přepínač,….)

127… příkaz nenalezen (nastavuje shell)

Příklady:

```
grep root  /etc/passwd; echo $?       # bude 0
grep rooot /etc/passwd; echo $?       # bude 1 (nejspis)
grep root  /etc/paswwd; echo $?       # bude 2 (nejspis)
echo $?                               # bude 0 (proc?)
grrp root  /etc/passwd; echo $?       # bude 127
```

Podmíněný příkaz

Neúplná podmínka:

```
if   PŘÍKAZ      # podmínkou je návratový kód  $?  PŘÍKAZu (0 = ano, jinak ne)
then             # klíčová slova musí být na začátku řádky nebo za středníkem
  příkazy        # je-li více příkazů na řádku, musí být odděleny středníkem
fi               # páruje se s if
```

Úplná podmínka:

```
if  PŘÍKAZ  ; then  příkazy ; else  příkazy ; fi
```

Podmínka if – elif (nahrazuje vnoření podmíněných příkazů)

```
if  PŘÍKAZ1  ; then  příkazy ; elif  PŘÍKAZ2  ; then  příkazy ; elif  PŘÍKAZ3
  then  příkazy ; else  příkazy ; fi
```

Příklad:

```
if [ $# -ne 1 ] ;then
    echo "Chyba, vola se: $0 <username>" >&2 ; exit 2
fi                                              # slo by i elif
if ! who|grep "$1" >/dev/null ;then             # ! = not
    echo "Uzivatel $1 neni prihlasen" >&2
else
    echo "Uzivatel $1 asi je prihlasen" >&2     # proc jen "asi" ☺
fi
```

Cykly

```
for PROM in SEZNAM ;do      # do proměnné PROM se postupně přiřazují
   PŘÍKAZ1                  # prvky (řetězce) ze SEZNAM
   PŘÍKAZ2; ……
done
```

```
while PRIKAZ ;do            # podmínkou je návratový kód příkazu PRIKAZ
     PŘÍKAZ1
     PŘÍKAZ2; ……
done
```

Příklady:

```
I=1
for SOUBOR in `ls` konec seznamu souboru ;do
    echo $I. Prvek seznamu je $SOUBOR
    ((I=I+1))
done
# dalsi priklad
[ "$I" -lt 1 ] && exit
I=1; VYSLEDEK=2
while [ $I -le "$I" ] ;do
    echo 2 na $I: $VYSLEDEK
    VYSLEDEK=`expr $VYSLEDEK \* 2` ; ((I++))
done
```

Příkaz case

```
case PROM  in          #  vyhodnotí se proměnná,
  vzor1)  příkaz1      #  její hodnota se postupně porovnává se vzory,
          příkaz2      #  nalezne-li se odpovídající,
          ……. ;;       #  provedou se příkazy za ním až k ";;"
  vzor2)  příkaz3      #  a pak se pokračuje příkazy za "esac"
          …….. ;;
  ……
esac
```

Příklad:

```
ZVIRE="$1"
case "$ZVIRE" in
  [Kk]oza|[Ss]lepice)              # "|" znamena "nebo"
    echo "$ZVIRE" je domaci zvire
    echo ":-)" ;;
  [Jj]elen|[Ss]rn*)
    echo "$ZVIRE" je divoke zvire;;
  holub|orel|slepice)              # "slepice" je tu
    echo "$ZVIRE" je ptak ;;       # zbytecne. Proc?
  *)echo Toto zvire neznam ;;
esac
```

Interaktivní skripty – příkaz read

read RADEK

read P1 P2 P3 ZBYTEK <soubor

Příkaz čte vstup až do konce řádku (je-li vstupem klávesnice, čeká na <Enter>), rozdělí jej pomocí $IFS a přiřadí do proměnných.

Příklady:

```
echo "Zadej cislo: \c"
read CISLO                    # spatne zadani (ne cislo) neresime
echo Druha mocnina $CISLO je $((CISLO*CISLO))

who | while read U X X X X ODKUD ;do          # muze ridit cyklus
        echo Uzivatel $U prihlasen z $ODKUD
      done

IFS=:
while read U X U_ID ZBYTEK ;do
  echo $U ma UID $U_ID    # proc se promenna jmenuje U_ID a ne UID?
done </etc/passwd
```

Pokročilá práce s proměnnými

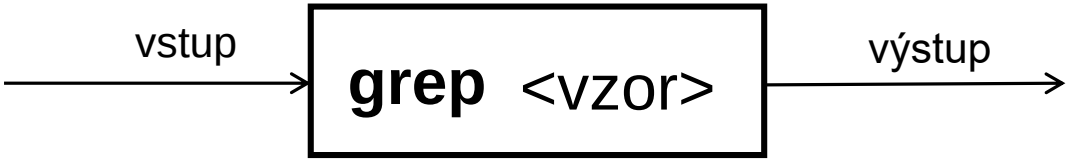
```
typeset -r PROM        # read only, nelze změnit ani smazat
typeset -i CISLO       # číselná proměnná
typeset -i16 HEXA      # číselná proměnná o základu 16  (jen ksh)
typeset -L10 VLEVO     # zarovnáno vlevo na 10 znaků (jen ksh)
typeset -R10 VPRAVO    # zarovnáno vpravo na 10 znaků (jen ksh)
typeset -l MALA        # všechna písmena malá (převádí se, ksh)
typeset -u VELKA       # všechna písmena velká (jen ksh)
```

Příklady:

```
#!/bin/ksh
read MALA                       # viz promenne definovane vyse
 VELKA=$MALA
 echo "Vse v malych: $MALA, vse ve velkych: $VELKA"
typeset -u -R15 FORMATED        # typy lze kombinovat
 FORMATED=$MALA
 echo "$FORMATED"               # bez uvozovek nemuze fungovat!!! Proc?
```

Upravte poslední příklad z předchozí stránky tak, aby výpis jmen a UID byl formátovaný (jména vlevo, UID vpravo) a setříděný podle UID.

Příkaz grep



- Textový filtr
- Na výstup kopíruje řádky obsahující zadaný vzor
- Vzor je regulární výraz
- Má řadu přepínačů, např.:
  - -i nerozlišuje malá a velká písmena
  - -v inverzní chování – kopíruje řádky **ne**obsahující vzor
  - -c vypíše pouze počet nalezených vzorů
  - -l vypíše pouze jméno souboru obsahující vzor

Příklady:

```
grep cat /usr/dict/words        # výpis slov obsahujících cat
                                # cat je konstantní regulární výraz
ls -l|grep Feb          # špatný (proč?) pokus vypsat soubory z února
grep -il "$USER" *      # seznam souborů obsahující jméno uživatele
prtconf|grep Memory     # výpis velikosti paměti (Solaris)
```

Regulární výrazy (Regular Expressions)

RE - formální prostředek pro popis textových řetězců

Základní regulární výraz podporuje následující (meta)znaky:

- **^** začátek řádky nebo doplněk množiny znaků **[^**
- **$** konec řádky
- **.** právě jeden libovolný znak
- ***** libovolný počet opakování předcházejícího znaku
- **[** začátek definice množiny znaků
- **]** konec definice množiny znaků
- **\<** začátek slova
- **\>** konec slova
- **** „ochrana“ následujícího znaku
- **\{** začátek definice opakovače
- **\}** konec definice opakovače

Příklad:

```
grep '^e.*[^aeiouy].$' /usr/dict/words  # výpis všech řádek
#       začínajících e, kde předposlední znak není samohláska
```

Příkaz egrep

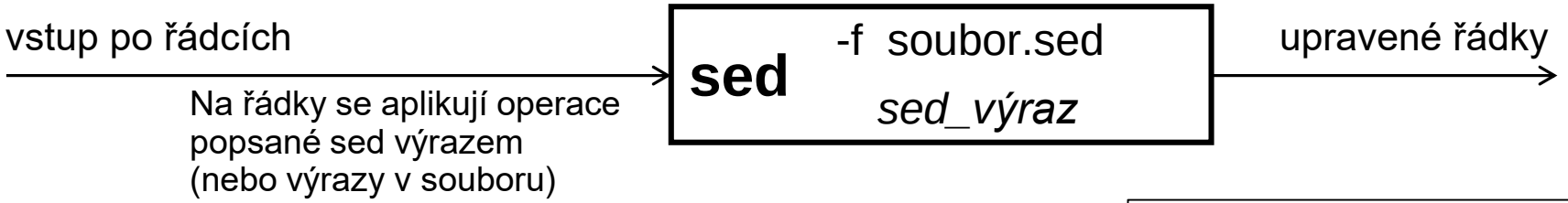
- Stejné chování jako **grep**
- Vzor může být definován **rozšířeným regulárním výrazem**
- Přepínače jsou podobné, jako u příkazu **grep**
- Nepodporuje znaky: **\(**, **\)**, **\n**, **\<**, **\>**, **\{**, **\}**
- Navíc ale podporuje znaky:

+	jeden nebo více výskytů předchozího znaku
?	žádný nebo jeden výskyt předchozího znaku
|	nebo
( )	označení regulárního podvýrazu

Příklady:

```
egrep ”^so+.$”  /usr/dict/words
egrep '^so?.$'  /usr/dict/words
egrep ”cat$|dog” /usr/dict/words
eqrep '(Sun|Mon|Tues)day' /usr/dict/words
```

SED – Stream EDitor



č	řádka číslo č
č1,č2	interval od č1 do č2
/RE/	řádka s reg. výrazem
$	poslední řádka
nic	všechny řádky

sed_výraz: [adresa]příkaz[parametry][přepínače]

soubor.sed:
(„sed skript“)

sed_výraz1
sed_výraz2
……atd……

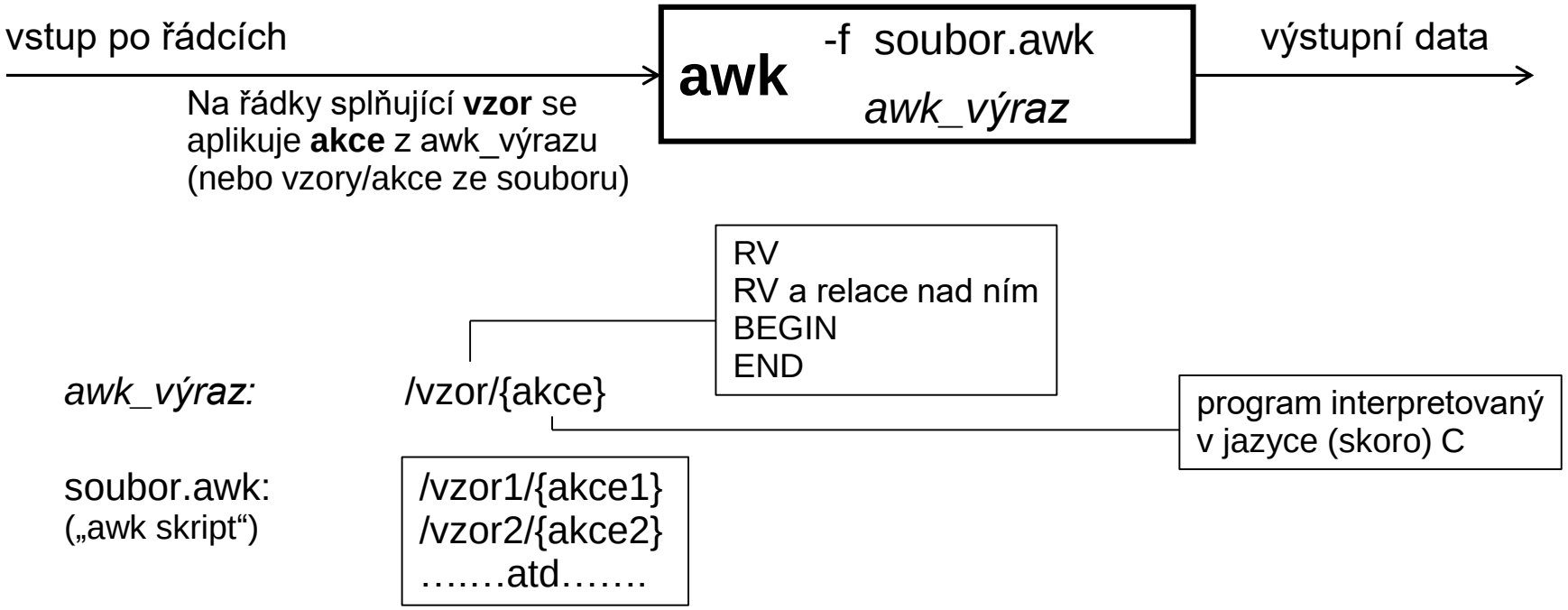
Příklady výrazů:

```
10d                 # zruší (vynechá) 10 řádku
/^[A-Z]/d           # zruší řádky začínající velkým písmenem
s/Jan/Honza/        # ve všech řádkách nahradí Jan za Honza
15,$a\              # za řádky od 15 do konce vloží řádku ********
********            # a (append) = za řádku,  i (insert) = před
```

Příklad použití:

```
ls -l / | cat –n | sed ´10,15d´ >upravena_data
```

Textový filtr **awk (nawk)**



Příklady výrazů:

```
/^.r/                          # vypíše řádky (default akce {print $0} ) splňující RV
{print $1 $3}                  # ze všech řádků se vypíše 1. a 3. pole
!/^d/{CNT++}                   # sečte počet řádek nezačínajících d
BEGIN{A=1; print “start“}      # inicialzace před zpracováním 1. řádky
END{print CNT, “neadresaru“}
```

Příklady:

```
awk '/.*cat.*/{cnt++;print $0};END{print cnt "x cat"}' /usr/dict/words
ls -l /etc | awk '/^-/{printf "%-30s ma delku: %10d\n",$9,$5'
```

Administrace Unixových systémů

Uživatel **root**

- administrátor – může vše
- v bezpečných systémech nyní implicitně jako role

Konfigurační soubory jsou většinou textové - příklady

- /etc/hosts
- /etc/resolv.conf
- /etc/passwd /etc/shadow /etc/group

Administrátorské nástroje – 3 kategorie, srovnání, výhody, nevýhody

- GUI (Graphical User Interface)
- CLI (Command Line Interface) - admin příkazy (useradd, ifconfig, lpadmin,...)
- Základní unixové příkazy (vi, mkdir, chown, cp, ln,..

Administrované subsystémy

- Uživatelé
- Disky, tiskárny, síťové karty
- Síťová administrace – NFS, Samba, LDAP
- Virtuální subsystémy – domény, zóny, XEN….

Administrace uživatelů

Stand alone stanice

Definováno v lokálních souborech (struktura viz dále):
/etc/passwd, /etc/shadow, /etc/group

Stanice v síti

Kombinace lokálních souborů a sdílené databáze:

- lokální soubory - systémoví uživatelé (proč? – viz dále)
- databáze LDAP, NIS - běžní uživatelé

Atributy uživatele

- username, UID
- skupina, GID - pravidla pro přidělování UID a GID
- heslo – pravidla a bezpečnost později
- poznámka, domovský adresář, login shell

- „systémoví“ uživatelé: bin, sys, lp, nobody, …..

Administrace uživatelů

Způsoby přidávání/rušení/modifikace uživatelů

- **GUI rozhraní**
  komfortní, menší znalosti, odolné proti chybám, nepraktické při větším počtu úprav (nelze automatizovat nebo omezeně)
- **CLI rozhraní** - řádkové administrátorské příkazy
  příkazy: useradd, usermod, userdel,
  mnoho přepínačů, flexibilnější, lze nad nimi vytvářet skripty a automatizovat
- **CLI rozhraní** – základní příkazy
  příkazy typu: cp, mkdir, vi, ...
  flexibilní, dovolí cokoli (včetně chyb), lze nad nimi vytvářet skripty

- **Přidání uživatele do databáze**
  krok navíc v případě stanice v síti

Hesla

Bezpečnostní pravidla

- zakódovaná hesla (hash) uložena v **/etc/shadow** -r--------
- “password aging”
- “síla” hesel **/etc/default/passwd** (Solaris)
- (ne)opakování hesel

Vytvoření obrazu hesla a kontrola

- zakódování hesla, sůl
- man –s 3c crypt (Solaris)
- char *crypt(const char *key, const char *salt)

Změna hash algoritmu (Solaris)

- adresář **/etc/security**
- soubory **crypt.conf** a **policy.conf**

$md5$salt$bflmpsvzkfoefjwj546erjheer87erenv122HIGH54Jhjhgf

Změna identity, příkaz su

su [-] [username [příkaz]]

- Mění všechny 3 identity uživatele na identitu „username“
- Je to suid program
- Neuvede-li se „–“, změní se pouze identita, prostředí ale zůstává (nebezpečné!)
- Uvede-li se „–“, je (téměř) ekvivalentní běžnému přihlášení
- Neuvede-li se „username“, použije se „root“
- Provádí-li ho root, nemusí zadávat heslo
- Lze použít i pro jednorázové provedení příkazu pod jinou identitou

Příkazy/Příklady

```
id - a                  # informace o identite uzivatele
su userx                # změna identit na userx (nebezpecne!)
su - userx              # změna identit a prostředí na userx
```

Příkaz sudo

`sudo [-u username] příkaz [argumenty]`

- Uživatel může jednorázově spustit příkaz pod cizí identitou „username“ bez znalosti hesla.
- Není-li identita zadána, spouští se pod root identitou
- Konfigurováno souborem ***/etc/sudoers***
- Při použití je třeba zadat heslo – vlastní! Po jistou dobu se pamatuje.

Výhody (+) a rizika (-)

- Delegace pouze určitých příkazů určitým uživatelům (+)
- Použití může být monitorováno (+)
- Konfiguraci (/etc/sudoers) lze sdílet různými systémy (+)
- Chyby v konfiguračním souboru (-)
- Zneužití neprivilegovaného účtu může způsobit „privilegovanou“ destrukci/útok (-)

Konfigurace sudo

Soubor `/etc/sudoers`

- Může číst i editovat pouze root
- Textový soubor, lze editovat přímo
- Nebo příkazem **`visudo`** – kontroluje syntaxi
  (editor lze zvolit nastavením proměnné **EDITOR**)
- "Standardní" syntaxe řádky ve **/etc/sudoers**:

kdo kde=(jako kdo) co

Pozn: Chybí-li pole **(jako kdo)**, je to totéž jako **(root)**

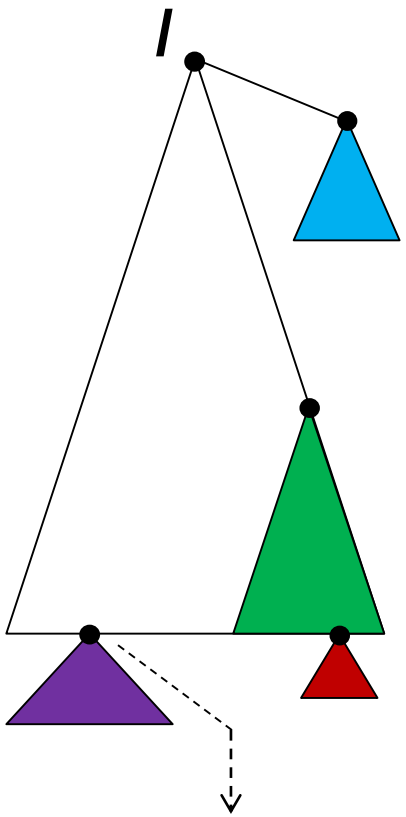
Příklady:

```
user1  host1=(root)  /usr/sbin/shutdown
user3 ALL=(user4,user5) /bin/kill,/bin/pkill
# lze definovat aliasy, viz např. ALL výše
```

Vztah logického a fyzického systému souborů

Logický systém souborů:

- Jediná stromová adresářová struktura
- Je tvořen fyzickými systémy souborů
- Je transparentní vůči fyzickým systémům souborů (mountovací bod je z jeho hlediska pouze adresář)
- Vrcholem je */*
- Pohyb příkazem cd



Fyzický systém souborů:

- Implementace souborů (adresářů) na fyzickém médiu
- Více či méně odpovídá „struktuře na úrovni metadat“ (i-node atd. - viz dříve), ale většinou ji navenek emuluje
- Mnoho různých typů – viz dále
- Udržuje (vytváří, připojuje,…) administrátor

Fyzické systémy souborů

3 kategorie

- **Diskové** – na discích, ale i na CD/DVD, flash paměti apod
- **Síťové** – připojovány vzdáleně pomocí různých protokolů (NFS, SMB/CIFS)
- **Virtuální** – v paměti, většinou systémově orientované, rychlé

Diskové

- Vytvářeny na celém disku, v partition, slice, někdy i v souboru
- Původní systém souborů v Unixu – S5
- Z něj se vyvinul UFS, je/byl ve všech variantách Unixu (Linux – ext[23]), postupně zdokonalován, mnoho výhod, ale neefektivní pro velké soubory
- Další: VxFS (optimalizován pro velké soubory), NTFS a FAT32/16 (MS Win),…
- ZFS – filesystém budoucnosti , vyvinut v Sun/Oracle pro Solaris, odstraňuje nevýhody a přináší mnoho principiálně nových vlastností – viz dále

Připojení do logického systému souborů (mimo ZFS, tam se udělá vše samo ☺)

1. Vytvořit fyzický FS — `newfs /dev/dsk/disk_name`
2. Vytvořit mountovací bod — `mkdir /data`
3. Připojit FS k mountovacímu bodu — `mount /dev/dsk/disk_name /data`

Virtuální disky

Problémy:

- Nemožnost vytvoření filesystému většího, než je fyzická kapacita disku
- Nemožnost vytvoření velkého počtu filesystémů na jednom disku
- Ztráta dat při poruše disku

Řešení (než přišel ZFS, tam je vše nativně ☺):

- Volume managery – SW, který nad fyzickými disky vytvoří virtuální disk požadovaných vlastností - Veritas, SVM, LVM,…
- HW řešení – disková pole, SW je součástí „firmware“, rychlejší

RAID (Redundant Array of Independent Disks)

- RAID0 – skládání (concatenation, striping)
- RAID1 – zrcadlení (RAID 0+1 a 1+0)
- RAID5 – diskové pole s paritou
- RAID6 – diskové pole s dvojitou paritou
- Soft slices – dělení disku na více částí
- RAID2, RAID3, RAID4 – existují, ale nepoužívají se (komplikované, pomalé)

Administrace disků

- Speciální soubory v **`/dev/dsk`** (blokové) a **`/dev/rdsk`** (znakové–raw)
  jméno **logické** (symbolický link) a **fyzické**
  velké a **malé** číslo zařízení
- Sektor, stopa, povrch, cylindr
- Fyzický, logický a lineární model disku

- Partition (PC): (p0) p1, p2, p3, p4 - Solaris
  (sda), sda0, sda1, sda2, sda3 - Linux
- Slice: s0, s1, … s7 – Solaris, na Sparc přímo, na PC v rámci partition
- Jmenná konvence (Solaris):
  c#t#d#s# např.: c0t1d0s5 (různé pro SCSI, IDE a SATA disky)
  c#d#p# např.: c1d1p0 (partition na PC)
- Definice:
  např. příkazy: **`format, fmthard`**
- Uloženo:
  partition tabulka (label, VTOC) – primární, sekundární (na PC)

Elektronický podpis I

- Využívá stejného principu, jako je ověření identity při ssh přihlášení
- Postaven na soukromém a veřejném klíči – tj. nesymetrické šifrování
- Klíč (certifikát) musí vydat tzv. certifikační agentura

Šifrování

- symetrická šifra
  - jeden klíč pro zašifrování i rozšifrování
  - rychlé, ale problém bezpečné distribuce klíčů, neprůkazné (kdo zašifroval)
- nesymetrická šifra
  - dva klíče (soukromý, veřejný), jedním se zašifruje, druhým rozšifruje
  - data zašifrovaná jedním klíčem nejdou rozšifrovat tím samým
  - soukromý klíč může mít „pin“ (passphrase)
  - pomalejší, ale bezpečnější, průkazné

Elektronický podpis II

Elektronický podpis garantuje

- **Autentičnost** dat (data odeslala příslušná osoba – ekvivalent běžného podpisu)
- **Konzistenci** podepsaných dat (data se během přenosu nezměnila)

Princip

Založen na nesymetrickém šifrování

Data se „zahashují“ privátním klíčem a jejich zakryptovaná forma je **elektronický podpis**, který se připojí ke zprávě.

Adresát ověří jeho pravost pomocí veřejného klíče

Nepopiratelnost

Autor nemůže popřít, že podpis nevytvořil

Garantuje tzv. „Certifikát“ – autorův (digitálně podepsaný) veřejný klíč

Pravost certifikátu je garantována tzv. „Certifikační agenturou“ (Důvěryhodná třetí strana), která klíče vydala → přenos důvěry

Důvěryhodnost Certifikační agentury garantuje Ministerstvo vnitra