1. cvičení (25.2. 2010)

Základní příkazy, ovládání shellu.

1. Ve svém domovském adresáři vytvořte adresář `ADR`
2. Do tohoto adresáře okopírujte soubory z adresáře `/usr/include`, jejichž názvy začínají znakem `m`, končí příponou `.h` a kromě toho v názvu obsahují číslici.
3. V adresáři `ADR` vytvořte podadresář `PODADRESAR`.
4. Prvních pět řádek z každého z okopírovaných souborů uložte do souboru `~/ADR/PODADRESAR/prvnichpet`.
5. Poslední řádky souborů v `~/ADR` uložte do souboru `~/ADR/PODADRESAR/posledni`.
6. Soubory v `~/ADR/PODADRESAR` spojte do jednoho souboru `~/ADR/PODADRESAR/prvniaposledni`.
7. Smažte soubory v `~/ADR/PODADRESAR` kromě `prvniaposledni`.
8. Do souboru `~/ADR/PODADRESAR/pocet` uložte počet souborů a adresářů v `~/ADR`. Nezapomeňte na soubory a adresáře, jejichž názvy začínají tečkou, ale nezapočítávejte „.“ a „..“.
9. Vypište dlouhé informace o adresáři `~/ADR/PODADRESAR`, ne jeho obsah.
10. Vymažte obsah souboru `~/ADR/PODADRESAR/prvniaposledni` bez toho, abyste smazali tento soubor.
11. Čas poslední modifikace souboru `~/ADR/PODADRESAR/pocet` nastavte na 13:30 1.2. 2009.
12. Čas poslední modifikace souboru `~/ADR/PODADRESAR/prvniaposledni` nastavte na stejný čas, jako má `/etc/passwd`.
13. Napište příkaz, který vypíše obsah adresáře `/usr/bin` podle jmen sestupně podle abecedy do souborů `~/ADR/PODADRESAR/bina` a `~/ADR/PODADRESAR/binb` a navíc jej vypíše na obrazovku stránkovaně (pomocí `less` nebo `more`)
14. Do souboru `~/ADR/PODADRESAR/prvniaposledni` přidejte řádku, jež obsahuje jen znak hvězdičky, tj *.
15. Smažte `~/ADR` i se všemi soubory a podadresáři v něm umístěnými.
16. Vypište počet skupin v systému (řádků v `/etc/group`)
17.

```
#!/bin/sh

# Ve svém domovském adresáři vytvořte adresář ADR
mkdir ~/ADR

# Do tohoto adresáře okopírujte soubory z adresáře
# /usr/include, jejichž názvy začínají znakem 'm', končí
# příponou '.h' a kromě toho v názvu obsahují číslici.
cp /usr/include/m*[0-9]*.h ~/ADR

# V adresáři ADR vytvořte podadresář PODADRESAR.
mkdir ~/ADR/PODADRESAR

# Prvních pět řádek z každého z okopírovaných souborů uložte do
# souboru ~/ADR/PODADRESAR/prvnichpet.
head -n 5 ~/ADR/m*[0-9]*.h >~/ADR/PODADRESAR/prvnichpet

# Poslední řádky souborů v ~/ADR uložte do souboru
# ~/ADR/PODADRESAR/posledni.
tail -n 1 ~/ADR/m*[0-9]*.h >~/ADR/PODADRESAR/posledni

# Soubory v ~/ADR/PODADRESAR spojte do jednoho souboru
# ~/ADR/PODADRESAR/prvniaposledni.
cat ~/ADR/PODADRESAR/* >~/ADR/PODADRESAR/prvniaposledni

# Smažte soubory v ~/ADR/PODADRESAR kromě prvniaposledni.
rm ~/ADR/PODADRESAR/prvnichpet
rm ~/ADR/PODADRESAR/posledni

# Do souboru ~/ADR/PODADRESAR/pocet uložte počet souborů a
# adresářů v ~/ADR.
ls ~/ADR | wc -l >~/ADR/PODADRESAR/pocet

# Vypište dlouhé informace o adresáři ~/ADR/PODADRESAR,
# ne jeho obsah.
ls -ld ~/ADR/PODADRESAR

# Vymažte obsah souboru ~/ADR/PODADRESAR/prvniaposledni bez
```

```
# toho, abyste smazali tento soubor.
echo >~/ADR/PODADRESAR

# Čas poslední modifikace souboru ~/ADR/PODADRESAR/pocet nastavte
# na 13:30 1.2. 2009.
touch -t 0902011330 ~/ADR/PODADRESAR/pocet

# Čas poslední modifikace souboru
# ~/ADR/PODADRESAR/prvniaposledni nastavte na stejný čas, jako
# má /etc/passwd.
touch -r /etc/passwd ~/ADR/PODADRESAR/prvniaposledni

# Napište příkaz, který vypíše obsah adresáře /usr/bin
# podle jmen sestupně podle abecedy do souborů
# ~/ADR/PODADRESAR/bina a ~/ADR/PODADRESAR/binb a
# navíc jej vypíše na obrazovku stránkovaně (pomocí less nebo more)
ls -r /usr/bin | tee ~/ADR/PODADRESAR/bina ~/ADR/PODADRESAR/binb | less

# Do souboru ~/ADR/PODADRESAR/prvniaposledni přidejte řádku, jež obsahuje
# jen znak hvězdičky, tj *.
echo '*' >>~/ADR/PODADRESAR/prvniaposledni

# Smažte ~/ADR i se všemi soubory a podadresáři v něm umístěnými.
rm -r ~/ADR

# Vypište počet skupin v systému (řádků v /etc/group)
wc -l /etc/group
```

18. Vypište řádky číslo 11-20 ze souboru `/etc/passwd`.

```
head -n20 /etc/passwd | tail -n10
```

19. Rozmyslete si, co se děje při následujících příkazech:
    1. `mv soubor /dev/null`
    2. `cp soubor /dev/null`
    3. `cat soubor >/dev/null`
20. Vypište deset největších souborů a pak deset nejmenších souborů v adresáři `/etc` `ls -s /etc | head` `ls -rs /etc | head`
21. V adresáři `/usr/bin` najděte soubor, který byl modifikován naposledy

```
ls -lrt /usr/bin | head -n 1
```

22. Spusťte si `vimtutor cs`

2. cvičení (4.3. 2010)

Základní příkazy, ovládání shellu.

Příkazy: `ssh`, `scp`, `w`, `who`, `who am i`, `id`, `mesg`, `talk`, `mail`, `last`.

3. cvičení (11.3. 2011)

Práva a jednoduché utility.

Příkazy: `chmod`, `chown`, `file`, `ln`, `diff`, `cut`, `paste`, `split`, `tr`

Pozor, na ulabech zkoušejte práva výhradně v adresáři `/tmp`, na ostatních je systém afs a práva tam fungují úplně jinak.

Příklady:

1. Vytvořte jednoduchý skript a přidělte mu práva pro spuštění.

```
cat >skript.sh <<EOF
#!/bin/sh

echo Ahoj světe
EOF

chmod a+x skript.sh
```

2. Vytvořte soubor, který může vlastník spustit, číst a psát do něj, členové skupiny jej mohou číst a psát do něj, ostatní jej mohou jen spustit.

```
echo Ahoj svete >/tmp/soubor
chmod 0761 /tmp/soubor

# Ekvivalentně například:
chmod u+rwx,g=u-x,o=u-rw /tmp/soubor

# nebo
chmod u=rwc,g=rw,o=x
```

3. Zkuste vytvořit skript, který nelze číst, ale má práva na spuštění, zkuste jej spustit. *Proč to u shellového skriptu nejde?*

```
cat >/tmp/skript.sh <<EOF
#!/bin/sh

echo Ahoj svete
EOF

chmod a+x-r /tmp/skript.sh
/tmp/skript.sh
```

4. Vytvořte adresář, do kterého může kdokoli přejít a přidávat soubory, ale jen vlastník smí vypsat obsah adresáře.

```
mkdir /tmp/adresar
chmod 0755 /tmp/adresar

#Případně ekvivalentně
chmod u+rwx,go=u-w /tmp/adresar
```

5. Vytvořte adresář, do něhož lze přejít, nelze číst jeho obsah a lze v něm přidávat soubory. Vytvořte v tomto adresáři soubor, k němuž budete mít práva jen ke čtení a zkuste ho smazat.

```
mkdir /tmp/adresar
chmod 0300 /tmp/adresar
# Případně ekvivalentně
chmod u+wx-r,go-rwx /tmp/adresar

touch /tmp/adresar/soubor
chmod 0400 /tmp/adresar/soubor
# Případně ekvivalentně
chmod u+r-wx,go-rwx /tmp/adresar/soubor

rm -f /tmp/adresar/soubor
```

6. Vypište celá jména uživatelů z /etc/passwd.

```
cut -d: -f4 </etc/passwd | sort > /tmp/uid
cut -d: -f3 </etc/group | sort | comm -23 - /tmp/uid
rm /tmp/uid
```

7. Z výstupu `ls -l` vyberte pouze 9 znaků s právy.

```
ls -l | cut -c2-10
```

8. Nahraďte v /etc/passwd g a G znakem @. Zjistěte potom pomocí diff, které řádky byly změněny.
   `tr gG @@ </etc/passwd | diff - /etc/passwd`
9. Prohoďte v /etc/passwd výskyty malého a velkého písmene „a“.

```
tr aA Aa </etc/passwd
```

10. Vypište dvojice *uid*:*login* za každého uživatele v /etc/passwd.

```
cut -d: -f3 /etc/passwd >/tmp/uid
cut -d: -f1 /etc/passwd | paste -d: /tmp/uid -
rm /tmp/uid
```

11. Předpokládejte, že máte tři soubory, `scitanec1`, `scitanec2` a `soucet`, každý z nich obsahuje stejný počet řádků s čísly. Sestavte tyto řádky do tvaru
    `scitanec1+scitanec2=soucet`

```
paste -d+= scitanec1 scitanec2 soucet
```

12. Vypište loginy ze souboru /etc/passwd tak, že na každém řádku bude <login malými písmeny>=<login velkými písmeny>

```
cut -d: -f1 </etc/passwd | tr "[:upper:]" "[:lower:]" >/tmp/maleloginy
cut -d: -f1 </etc/passwd | tr "[:lower:]" "[:upper:]" | paste -d= /tmp/maleloginy -
rm /tmp/maleloginy
```

13. Vypište loginy ze souboru /etc/passwd po pěti na řádek, na každém řádku jsou oddělené čárkami.

```
cut -d: -f1 </etc/passwd | paste -d"," - - - - -

# Alternativně a ekvivalentně lze:
cut -d: -f1 </etc/passwd | paste -s -d",,,,\n" -
```

14. Rozdělte soubor na kusy po pěti řádcích a pak jej zase spojte do jednoho souboru.

```
# Budeme dělit soubor /etc/passwd
split -l5 /etc/passwd kusypasswd
# Spojený soubor uložíme do aktuálního adresáře, protože k zápisu do
# /etc nemáma práva
cat kusypasswd* >passwd
rm kusypasswd*
```

15. Vypište loginy ze souboru /etc/passwd do deseti řádků, na každém řádku jsou loginy oddělené mezerami. (Řešení není)

4. cvičení (18.3. 2011)

Příkazy: `sort`, `comm`, `join`.

1. Vypište loginy uživatelů z /etc/passwd utříděné sestupně lexikograficky.

```
sort -t: -k1,1r /etc/passwd | cut -d: -f1
```

2. Vypište jména skupin setříděná podle gid.

```
sort -t: -k3,3n /etc/group | cut -d: -f1
```

3. Vypište počet různých shellů v /etc/passwd.

```
sort -t: -u -k7,7 /etc/passwd | wc -l
```

4. Vypište jméno skupiny s 5. nejvyšším gid.

```
sort -t: -k3,3nr /etc/group | head -n 5 | tail -n 1 | cut -d: -f1
```

5. Vypište názvy souborů, které se nacházejí `/usr/bin` i v `/bin`.

```
ls /usr/bin | sort >/tmp/usr_bin
ls /bin | sort | comm -12 - /tmp/usr_bin
rm /tmp/usr_bin
```

6. Vypište čísla skupin, která jsou v /etc/group, ale nejsou použita v /etc/passwd.

```
cut -d: -f4 </etc/passwd | sort > /tmp/uid
cut -d: -f3 </etc/group | sort | comm -23 - /tmp/uid
rm /tmp/uid
```

5. cvičení (25.3. 2011)

Příkazy: `xargs`, `join`

1. Zkuste rozdíl mezi `echo $PATH; echo "$PATH"; echo '$PATH';`
2. Zkuste rozdíl mezi `echo *; echo "*"; echo '*';`
3. Vypište dlouhý výpis adresářů obsažených v `$PATH`, lépe řečeno dlouhé informace o nich, ne jejich obsah.

```
echo $PATH | tr ":" " " | xargs ls -ld
```

4. Vytvořte soubor, který obsahuje 3 řádky:

```
a b c
d e f
g h i
```

   a na něm zkuste následující příkazy:

```
cat soubor | xargs echo
cat soubor | xargs echo -
cat soubor | xargs -n 2 echo -
cat soubor | xargs -I{} echo "-{}-" "-{}-"
cat soubor | xargs -I{} -n 2 echo "-{}-" "-{}-"
```

5. Přejmenujte všechny soubory v daném adresáři na soubory téhož jména psané velkými písmeny. Tj. například ze "skript.sh" by vznikl soubor "SKRIPT.SH".

```
ls | tee malymi | tr "[:lower:]" "[:upper:]" >velkymi
xargs -I{} echo '"{}"' <malymi >malymi2
xargs -I{} echo '"{}"' <velkymi >velkymi2
paste -d" " malymi2 velkymi2 | xargs -n 2 mv
rm malymi malymi2 velkymi velkymi2
```

6. Rozdělte soubor /etc/passwd na části po pěti řádcích a potom tyto části poskládejte v opačném pořadí do jiného souboru, (to jest nejprve jde posledních pět řádků v pořadí jako v /etc/passwd, potom předposledních pět atd

```
split -l5 /etc/passwd kusypasswd
ls -r kusypasswd* | xargs cat >passwd
rm kusypasswd*
```

7. Vypište jméno skupiny, která má v souboru /etc/group uvedených nejvíc členů. Vypište i počet těchto členů.

```
cut -d":" -f4 /etc/group | tr -sc ",\n" "-" | tr -d "," >/tmp/poctyskupin
cut -d":" -f1 /etc/group | paste -d" " - /tmp/poctyskupin |\
   sort -k2,2r | head -n 1 >/tmp/skupina
cut -d" " -f2 /tmp/skupina | tr -d -c "-" | wc -c |\
   paste -d" " /tmp/skupina - | cut -d" " -f1,3

rm /tmp/poctyskupin /tmp/skupina
```

8. Vypište dvojice oddělené dvojtečkou, kde v prvním poli je login uživatele a v druhém je jméno jeho primární skupiny. K vygenerování seznamu použijte příkaz join na vhodně předupravené soubory `/etc/passwd` a `/etc/group`

```
cut -d: -f1,3 /etc/group | sort -t: -k2,2 >skupiny
cut -d: -f1,4 /etc/passwd |sort -t: -k2,2 | join -1 2 -2 2 -t: -o 1.1,2.1 - skupiny
rm skupiny
```

6. cvičení (1.4. 2011)

Příkazy: `find`, `grep`

Pozor, na ulabech zkoušejte `find` raději jen v lokálních adresářích jako `/tmp, /usr, /etc` a podobně, ale ne ve vašem domovském adresáři, neboť ten je uložen na filesystému afs a s tím si `find` nerozumí (nebo to tak bylo v minulých letech a nevím o tom, že by se na tom něco změnilo).

1. Napište příkaz, který vypíše všechny (pro vás viditelné) soubory nebo adresáře, které mají ve jménu podřetězec 'bin'.

```
find . -name "*bin*" "(" -type d -o -type f ")
```

2. Napište příkaz, který vypíše počet všech (pro vás viditelných) adresářů v podstromu /etc.

```
find /etc -type d | wc -l
```

3. Vypište všechny symbolické linky z adresáře /etc, nezahrnujte celý podstrom, jen to, co je přímo v adresáři /etc.

```
find /etc -type l '!' -path "/etc/*/*"
```

4. Z podstromu adresáře /usr/bin vypište soubory, na které ukazují alespoň tři hardlinky. `find /usr/bin -links +3`
5. V podstromu adresáře /tmp najděte všechny soubory, které jsou větší než sto kilobyte a jsou čitelné pro všechny.

```
find /tmp -size +102400c -perm -a+r
```

6. Příkaz, který smaže soubory jmenující se "core", které jsou staré alespoň pět dní. (Příslušné soubory si vytvořte sami v adresáři /tmp, abyste si to mohli zkusit.)

```
find . -name "core" -mtime +5 -exec rm -f {} ";"

# Ekvivalentně pomocí xargs:
find . -name "core" -mtime +5 | xargs rm -f
```

7. Napište mírně zjednodušenou verzi toho, co dělá `which`. Jmenovitě: Popořadě v každém adresáři zadaném v proměnné `PATH` hledejte daný soubor (vyberte si nějaký). Tento soubor nesmí být v podadresáři daného adresáře, ale jen přímo v něm. Jde o soubor nebo symbolický link (ne adresář). Buď jste vlastník souboru a soubor je čitelný a spustitelný pro vlastníka, nebo je soubor čitelný a spustitelný pro všechny (tedy uživatele, skupinu i ostatní, to je to zjednodušení, protože nemusíte testovat, jestli patříte do stejné skupiny jako soubor a soubor je spustitelný pro skupinu).

```
echo $PATH | tr ":" "\n" | xargs -I{}\
   find {} "(" -path "{}" -o -prune ")"\
      -path "{}/find" "(" -type f -o -type l ")"\
      "(" "(" -user "$LOGNAME" -a -perm -u+rx ")" -o -perm -a+rx ")" |\
   head -n 1
```

8. Příkaz, který vypíše všechny soubory a adresáře patřící skupině vmware nebo stunnel v podstromu adresáře /etc, které jsou spustitelné pro vlastníka, skupinu i všechny ostatní nebo do kterých můžou členové skupiny zapisovat.

```
find /etc "(" -group vmvware -o -group stunnel ")" "(" -perm -a+x -o -perm -g+w ")"
```

9. Příkaz, který pro každý soubor se jménem "core" v podstromu adresáře /tmp vypíše jeho cestu a velikost v bytech. Na každém řádku výstupu by mělo být: cesta/soubor velikost.

```
find /tmp -name "core" | tee /tmp/coresoubory | xargs -I{} ls -ld "{}" ";" |\
   tr -s " " " " " " | cut -d" " -f5 >/tmp/velikosti
paste -d" " /tmp/coresoubory /tmp/velikosti
rm /tmp/velikosti /tmp/coresoubory
```

10. Ze souboru `/etc/passwd` vyberte řádky, na nichž je login s alespoň třemi znaky.

```
grep '^[^:]\{3\}' /etc/passwd
```

11. Ze souboru `/etc/passwd` vyberte řádky, na nichž login začíná samohláskou.

```
grep '^[aeiouy]' /etc/passwd
```

12. Ze souboru `/etc/passwd` vyberte řádky s loginy, které obsahují číslici.

```
grep '^[^:]*[[:digit:]]' /etc/passwd
```

13. Z výpisu `getent passwd` vyberte řádky, kde položka s domovským adresářem začíná řetězcem `/afs`.

```
getent passwd | grep ':/afs.*:[^:]*$'
```

14. Z výpisu `getent passwd` vyberte řádky s loginy, které se skládají z osmi znaků, z nichž první čtyři jsou malá písmenka anglické abecedy, a poslední čtyři jsou číslice.

```
getent passwd | grep '^[[:lower:]]\{4\}[[:digit:]]\{4\}:'
```

15. Z výpisu `ls -l` vyberte řádky s adresáři (prvním znakem na řádku je `d`).

```
ls -l | grep '^d'
```

16. Z výpisu `ls /usr/bin` vyberte soubory, jejichž názvy neobsahují číslici.

```
ls /usr/bin | grep '^[^[:digit:]]*$'
```

7. cvičení (8.4. 2011)

Příkazy: `grep`, `sed`, `ed`

□ Ze souboru /etc/passwd vypište řádky uživatelů, jejichž domovský adresář obsahuje jako podřetězec jejich login.

□ Pomocí sedu nahraďte ve výstupu `getent passwd` domovské adresáře `/afs/ms/u/X/Xxxx` na `/home/Xxxx` a naopak. X označuje první znak loginu, Xxxx označuje login. *(Každý směr zvlášť.)*

```
# /afs/... -> /home/...
getent passwd | sed 's%:/afs/ms/u/./\([^:]*:[^:]*\)$%:/home/ %' >passwd2

# /home/... -> /afs/...
cat passwd2 | sed 's%:/home/\(\([^/:]\)[^/:]*:[^:]*\)$%:/afs/ms/u// %' >passwd3

# Čistě pro kontrolu ...
getent passwd | diff - passwd3
```

□ Ze souboru /etc/passwd vypište jen sudé řádky. (*Pro kontrolu použijte `cat -n`, ačkoli nejde o standardní parametr.*)

```
cat -n /etc/passwd | sed -n 'n; p'
```

□ Před první řádku vstupního souboru vložte řádku s `#!/bin/sh`.

```
#!/usr/bin/sed -f
# Toto je skript pro sed, použití pomocí sed -f

1i\
#!/bin/sh
```

□ Napište skript pro sed, který odstraní ze shellového skriptu komentáře, přičemž

1. Pokud jde o první řádku s formátem `^#!.*`, pak tento komentář neodstraňujte.
2. Řádky, kde před `#` není nic než mezery, odstraňte úplně.
3. Pokud se před `#` vyskytují nějaké jiné znaky, odstraňte z řádky vše od znaku `#` (včetně) do konce řádky.

```
#!/usr/bin/sed -f
# Toto je skript pro sed, použití pomocí sed -f

# Přeskočení první řádky, pokud jde o komentář typu #!
1{
s/^#!/&/
t
}

# Smazání řádek, kde před # nic není
/^[[:space:]]*#/d

# Odstranění komentáře z ostatních řádek (ignoruje fakt, že # se může
# vyskytovat i v rámci řetězce v uvozovkách a pod., kdybychom toto
# chtěli kontrolovat, bylo by to výrazně komplikovanější).
s/#.*$//
```

□ Na začátky sudých řádek souboru /etc/passwd vložte znak „l“, na začátky lichých řádek vložte znak „s“.

```
#!/usr/bin/sed -f
# Toto je skript pro sed, použití pomocí sed -f

s/^/l/
n
s/^/s/
```

8. cvičení (15.4. 2011)

Příkazy: `sed`, `ed`

1. Na začátky desáté až dvacáté řádky souboru /etc/passwd přidejte znak `#`.

```
#!/bin/sh

sed '10,20s/^/#/' /etc/passwd
```

2. Mezi každé dvě řádky souboru /etc/passwd vložte prázdnou řádku.
(řešení v sedu)

```
#!/bin/sh

# $b zařídí, že se nebude nic přidávat za poslední řádku (při
# poslední řádce se skočí na konec skriptu bez vykonání příkazu a).
# Příkaz a pak provádí samotné přidání nové řádky za aktuální.

sed '$b; a\
' /etc/passwd

# Ekvivalentně lze i pomocí i (adresace zabezpečuje, že se nic nepřidá
# před první řádku):

sed '2,$i\
' /etc/passwd
```

3. Předpokládejte, že řádky vstupního souboru obsahují jen znaky '(' a ')', vytvořte skript pro sed, který zkontroluje, zda je řetězec na řádce správně uzávorkovaný. Pokud ano, nahradí řádku řetězcem 'ano', v opačném případě řetězcem 'ne'.

```
#!/usr/bin/sed -f

# Toto je skript pro sed, použití pomocí sed -f

# Skript opakovaně hledá výskyty podřetězce (), (pokud je jich v
# aktuálním řetězci víc, může je zpracovat najednou, proto je použito
# g u příkazu s) a odstraňuje je, pokud už se žádný takový podřetězec
# nevyskytuje a přesto ještě není řetězec prázdný, pak nejde o správné
# uzávorkování.

: zacatek
s/()//g
t zacatek
/./s/.*/ne/
/^$/s/^/ano/
```

4. Pomocí sedu vložte na začátek vstupního souboru řetězec #!/bin/bash a zakomentujte všechny bloky řádků tvaru

```
/*
...
komentar
```

```
... */
```

to jest na začátky řádků přidejte znak #. Předpokládejte, že začátek a konec komentáře jsou každý na jiném řádku a že před začátkem komentáře případně za koncem komentáře už na témže řádku nic není.

```
#!/usr/bin/sed -f

# Toto je skript pro sed, použití pomocí sed -f

1i\
#!/bin/bash
/\/\*/,/\*\/s%^%#%
s%/\*%%
s%\*/%%
```

5. Některé z předchozích příkladů zkuste udělat i v edu. (Řešení viz u každého příkladu).

9. cvičení (22.4. 2011)

Příkazy: `read`, řídící struktury (`for`, `while`, `if`, `case`, a pod.), `printf`, `expr`, `test`.

Další: Proměnné a expanze, přesměrování, poziční a speciální parametry ($#, $0, $n, ${n}, shift, "$@", set -, $?, $$), kombinace příkazů pomocí &&, složený příkaz a vyvolání subshellu, zpětné apostrofy.

Příklady:

1. Vytvořte skript, který dostane n+1 parametrů, první obsahuje číslo x mezi 1 a n, skript vypíše parametr s číslem (x+1). Tj. například, `skript 2 a b c` vypíše `b`, tedy ten třetí parametr (ale druhý, když budete počítat od `a`). (*Použijte* `shift x`.)

```
#!/bin/sh

shift $1
echo $1
```

2. Vytvořte skript, který dostane dva parametry, název souboru a řetězec, skript vypíše "ano", pokud se řetězec nachází v daném souboru a "ne" pokud se v něm nenachází. Jaký je význam návratového kódu příkazu `grep`?.

```
#!/bin/sh

grep "$2" "$1" >/dev/null 2>&1
echo $? | sed 's/0/ano/; s/1/ne/'
```

3. Vytvořte skript, který skript dostane (n+1) parametrů, první parametr je název souboru a ostatní obsahují řetězce, které neobsahují mezeru. Pro každý řetězec ověřte, jestli je obsažen v souboru zadaném v prvním parametru, pokud ano, vypište za něj řádku:
řetězec `ano`
V opačném případě vypište:
řetězec `ne`

```
#!/bin/sh
soubor="$1"
shift
for retezec in "$@"
do
   if grep -F "$retezec" "$soubor"
```

```
        then
            echo $retezec ano
        else
            echo $retezec ne
        fi
    done
```

4. Co vypíší následující příkazy:

```
a="ahoj"; { echo $a ; a="cau" ; echo $a ; } ; echo $a
b="ahoj"; ( echo $b ; b="cau" ; echo $b ; ) ; echo $b
x="ahoj"; x="cau" sh -c 'echo $x'; echo $x
y="ahoj"; y="cau"; sh -c 'echo $y'; echo $y
z="ahoj"; sh -c 'echo $z'
export u="ahoj"; sh -c 'echo $u'
```

5. Rozmyslete si, co dělá následující příkaz:

```
a=1;b=2; { a=LEVY; echo $a >&2; } |\
{ b=PRAVY; echo $b; tr 'A-Z' 'a-z';}; echo "A=$a,B=$b"
```

   a jak se liší od

```
a=1;b=2; { a=LEVY; echo $a; } |\
{ b=PRAVY; echo $b; tr 'A-Z' 'a-z';}; echo "A=$a,B=$b"
```

6. Rozmyslete si, co dělá následující příkaz, co se vypíše do out a co do err a proč.

```
{ { { echo 1aaa;echo 2bbb >&2;} 3>&1 1>&2 2>&3 |\
{ tr 'a-z' 'A-Z'; echo 2tr >&2;} 2>&1; } \
3>&1 1>&2 2>&3; }>out 2>err
```

7. Jaký je exit status následujících dvou příkazů?

```
{ { { cmd1; echo $? >&3; } |\
cmd2 >&4; } 3>&1 | { read e; exit $e; }; } 4>&1;

{ { { cmd1; chyba=$?; } |\
cmd2 >&4; } 3>&1 | { exit $chyba; }; } 4>&1;
```

8.Napište skript, který dělá totéž, co `cp`, ale s opačným pořadím parametrů (cíl je jako první)

```
#!/bin/sh
cil="$1"

shift
cp "$@" "$1"
```

10. cvičení (29.4. 2011)

Příkazy: `printf`, `expr`, `test`, `ps`, `kill`, `trap`, `eval`.

Další: Signály a procesy.

Příklady:

1. Napište skript, který dostane tři parametry, všechny číselné, a vypíše na standardní výstup čísla oddělená mezerami počínaje $1, konče $2 a s krokem $3. Pokud skript dostane jen dva parametry, pak se implicitně použije krok 1.

```
#!/bin/sh

if [ $# -lt 2 ] || [ $# -gt 3 ]
then
   echo Špatný počet parametrů
   exit 1
fi

if [ $# -eq 3 ]
then
   krok=$3
else
   krok=1
fi

x=$1
while [ $x -le $2 ]
do
   echo $x
   x=`expr $x + $krok`
done
```

2. Totéž jako předchozí příklad, ale čísla budou každé na zvláštní řádce a budou zarovnaná na počet míst daný délkou nejdelšího čísla (tj. délkou $2). Čísla budou zarovnaná nulami na začátku. Tj. například
   `skript 0 10 2`
   by vypsal:

```
00
02
04
06
08
10
```

```
#!/bin/sh
if [ $# -lt 2 ] || [ $# -gt 3 ]
then
   echo Špatný počet parametrů
   exit 1
fi

if [ $# -eq 3 ]
then
   krok=$3
else
   krok=1
fi

x=$1
delka=${#2}
while [ $x -le $2 ]
do
   printf "%0${delka}d\n" $x
   x=`expr $x + $krok`
done
```

9. Napište skript, který přejmenuje všechny soubory v aktuálním adresáři na soubory se stejným jménem, kde jsou velká písmena změněná na malá. (*Například soubor 'ZALOHA.ZIP' by přejmenoval na 'zaloha.zip'*)

```
#!/bin/sh

for x in *
do
   echo mv $x `echo $x | tr '[:upper:]' '[:lower:]'`
done
```

10. Napište skript, který přejmenuje všechny soubory v aktuálním adresáři s příponou jpg na soubory s toutéž příponou ale s čísly místo názvů, kde čísla budou zarovnaná na tolik míst, kolik je třeba pomocí nul. (Využijte toho, co jste už udělali v jednom z předchozích příkladů.) Tj. například pokud jsou v adresáři soubory alice.jpg, bob.jpg cyril.jpg a daniel.jpg, přejmenoval by je popořadě skript na 1.jpg, 2.jpg, 3.jpg a 4.jpg. Pokud by těchto souborů bylo alespoň deset ale méně než 100, byly by pojmenovány jako 01.jpg, 02.jpg, 03.jpg, 04.jpg, ...

```
#!/bin/sh
# Pokud už tam náhodou je nějaký soubor pojmenovaný číslem výsledného
# formátu, tak to může dělat problémy, proto to případně zkoušejte s
# přesunováním do nějakého prázdného adresáře.

x=1
pocet=`ls *.jpg | wc -l`
for soubor in *.jpg
do
   mv $soubor `printf "%0${#pocet}d\n" $x`.jpg
   x=`expr $x + 1`
done
```

11. Napište skript, který na začátek každé řádky vstupního souboru přidá číslo řádky. Tj. to, co dělá cat -n, ale protože -n je nestandardní parametr, nelze jej použít. (Lze vyřešit triviálně použitím příkazu n edu, zkuste tedy i bez použití edu pomocí shellových příkazů.)

```
#!/bin/sh
# Nejprve triviální řešení v edu (například pro /etc/passwd)
ed /etc/passwd <<\KONEC
%n
Q
KONEC

# Řešení v shellu bez edu (například pro /etc/passwd)
x=1
while read radek
do
   echo $x $radek
   x=$((x + 1))
done </etc/passwd
```

12. Spočítá následující skript správně počet řádků souboru /etc/passwd?

```
pocet=0;
cat /etc/passwd | while read x
do
   pocet=`expr $pocet + 1`
done
echo $pocet
```

A co následující skript?

```
cat /etc/passwd | {
   pocet=0
   while read x
   do
      pocet=`expr $pocet + 1`
   done
   echo $pocet
}
```

A co následující skript?

```
pocet=0
while read x
do
   pocet=`expr $pocet + 1`
```

```
done </etc/passwd
echo $pocet
```

A konečně co následující skript?

```
pocet=0
while read x </etc/passwd
do
   pocet=`expr $pocet + 1`
done
echo $pocet
```

Jaký je mezi těmito skripty rozdíl?

11. cvičení (6.5. 2010)

Příkaz: `awk`.

Příklady:

1. Napište awk skript, který vypíše každou desátou řádku vstupu.

```
#!/bin/awk -f

(NR % 10) == 0
```

2. Napište awk skript, který otočí pořadí slov na každé řádce vstupu (napíše je v opačném pořadí)

```
#!/bin/awk -f

{
   for (pole = NF; pole > 1; pole --)
   {
      printf ("%s" OFS, $pole);
   }
   printf ("%s" ORS, $1);
}
```

3. Napište awk skript, který očísluje řádky na vstupu a vypíše je i s čísly (jako `cat -n`).

```
#!/bin/awk -f

{
   print NR " " $0;
}
```

4. Napište awk skript, který vypíše tolik náhodných čísel mezi 0 a 1, kolik dostane zadáno parametrem.

```
#!/bin/sh
# Toto je skript, který obaluje awk skript
# Sám dostane počet potřebných parametrů ve svém parametru.

awk -v pocet="$2" 'BEGIN {
   srand();
   for (i=0; i<pocet; ++i)
   {
      print rand ();
   }
}'
```

5. Napište awk skript, který ze souboru /etc/passwd (výstupu `getent passwd`) vypíše plná jména uživatelů, jejichž login má tvar čtyři písmena následovaná čtyřmi číslicemi.

```
#!/bin/awk -f

BEGIN {
   FS = ":";
}

# V awk standardně nefungují opakovátka, je nutno vypsat tam
# podřetězec vícekrát.
$1 ~ /^[[:alpha:]][[:alpha:]][[:alpha:]][[:alpha:]][0-9][0-9][0-9][0-9]$/ {
   print $5;
}
```

6. Napište awk skript, který vypíše počet linků, tedy tagů <a ve vstupním html souboru.

```
#!/bin/awk -f
# Jako RS zvolíme "<", protože podle toho se dělí HTML soubor
# nejvhodněji. Specifikace ani nepřipouští víceznakové RS (v tom
# případě je výsledek nedefinovaný), regexp může být jen ve FS, tak to
# ani jinak nejde.
BEGIN {
   RS = "<";
   pocetAcek = 0;
}

($1 ~ /^a$/) && ($2 ~ /^href($|=)/) {
   print
   pocetAcek ++;
}

END {
   print pocetAcek;
}
```

7. Napište awk skript, který v textu zakomentuje všechny odstavce, které začínají komentářovým řádkem. (To jest, před druhou a další řádky tohoto odstavce předřadí komentářový znak.) Znak uvozující komentář je '#'. Odstavce se oddělují prázdným řádkem.

```
#!/bin/awk -f
# Láká sice použít RS="\n\n", ale RS nemůže být víceznakové, a tak to
# tak jednoduše nejde.

BEGIN {
   novyodstavec = 1;
   komentovat = 0;
}

novyodstavec && $1 ~ /^#/ {
   komentovat = 1;
}

! novyodstavec && NF > 0 && komentovat {
   $1 = "#" $1
}

NF > 0 {
   novyodstavec = 0;
}

NF == 0 {
   novyodstavec = 1;
   komentovat = 0;
}

{ print }
```

12. cvičení (13.5. 2010)

Příkazy: `eval`, `kill`, `ps`, `trap`, `sleep`

Příklady:

1. Napište awk skript, který vypíše posledních 10 řádků souboru.

```
#!/bin/awk -f

BEGIN {
   konecBufferu = 0;
}

{
   buffer [konecBufferu] = $0;
   konecBufferu = (konecBufferu + 1) % 10;
}

END {
   if (NR > 0)
   {
      if (NR < 10)
      {
         indexVBufferu = 0;
      }
      else
      {
         indexVBufferu = konecBufferu;
      }
      do
      {
         print buffer [indexVBufferu];
         indexVBufferu = (indexVBufferu + 1) % 10;
      }
      while (indexVBufferu != konecBufferu)
   }
}
```

2. Napište awk skript, který dostane na vstupu html soubor a vypíše adresy, na které se tento soubor odkazuje (to jest za značku <a href="adresa"> vypíše adresa). Pozor na to, že ta značka může být rozložena přes víc řádků a můžou tam být kolem `a`, `href` mezery. Volbou vhodného RS a FS si výrazně usnadníte práci.

```
#!/bin/awk -f

BEGIN {
   RS = "<";
   FS = "[[:space:]]*\"[[:space:]]*";
}

$1 ~ /^[[:space:]]*a[[:space:]]+href[[:space:]]*=[[:space:]]*/ {
   print $2
}
```

3. Napište skript, který bude očekávat vstup na standardním vstupu, načtený vstup pouze zopakuje na svůj standardní výstup. Pokud skript dostane signál SIGKILL (9), tak samozřejmě skončí, ale pokud dostane signál SIGTERM (15), SIGQUIT (3), či SIGINT (2) (to je ten, který je poslán pomocí Ctrl-c), neskončí a jen vypíše číslo toho signálu, který takto obdržel. Při SIGINT navíc vypíše naposledy vypsanou řádku vstupu.

```
#!/bin/sh

trap "echo 15" TERM
trap "echo 3" QUIT
trap 'echo 2; echo $x' INT

while read x
do
   echo $x
done
```

4. Napište skript, který dostane jeden číselný parametr s počtem sekund. Skript potom počká zadaný počet sekund a skončí. Signály Pokud skript obdrží signál SIGINT (2 - Ctrl-C), tak vypíše, kolik sekund už od začátku uběhlo. Pokud obdrží signál SIGQUIT (3 - Ctrl-\), tak vypíše, kolik ještě zbývá sekund do konce, ale svůj běh nezastaví. Pokud skript obdrží signál SIGTERM, vypíše, kolik uběhlo sekund od začátku, kolik zbývá do konce a skončí. Návratová hodnota skriptu je buď 0, pokud skončil ve správný čas, nebo zbývající počet sekund, pokud skončil kvůli SIGTERM.
5. Napište skript, který dostane jako parametr název souboru, skript očekává poté na standardním vstupu posloupnost čísel, každé je na nové řádce a je indexem řádky, kterou má skript vypsat. Skript implementujte tak, že nejprve načte řádky souboru do pole, které implementujete pomocí eval (tj. `eval '$'POLE_$INDEX`), poté dotazy zodpovídejte přístupem do tohoto "pole".
6. Napište skript, který očekává na standardním vstupu řádky ve tvaru:
   a+*b*=*c*
   kde *a*, *b* a *c* jsou čísla, skript načte pro každou řádku tato tři čísla a zkontroluje, jestli je rovnost správná. Pokud není správná, ohlásí chybu s číslem řádky, na které k ní došlo. Na závěr vypíše celkový počet chyb. Použijte shellový while cyklus s `read a b c` a vhodnou volbou IFS (+=).

13. cvičení (20.5. 2010)

Příklady:

1. Napište skript, který dostane na vstupu dva parametry, čas ve formátu [+][[HH:]MM:]SS (místo chybějících časových údajů se doplní nuly) a příkaz. Skript počká do zadaného času (bez ohledu na den) a v něm vykoná příkaz (ten může obsahovat i expanzi - použijte eval). Pokud je čas uvozen znakem +, pak to znamená, že se má čekat po zadaný čas a po něm příkaz vykonat. Tj.
   ```
   skript 15:42:00 'echo ahoj'
   ```
   vypíše v 15:42 řetězec ahoj, zatímco příkaz
   ```
   skript +2:30 'echo ahoj'
   ```
   vypíše řetězec ahoj po dvou a půl minutách od spuštění. (4b)
2. Napište skript, který dostane na vstupu seznam souborů, poté čte ze standardního vstupu dvojice řetězců, ve dvojici jsou řetězce oddělené mezerou a každá dvojice je na zvláštní řádce. Jde-li o dvojici s1 s2, pak provede náhradu řetězce s1 za s2 ve všech souborech zadaných v parametru. (Můžete předpokládat, že v řetězcích se nevyskytují žádné divné znaky, které by se pletly do regulárního výrazu.) (4b)
3. Napište skript, který dostane signál a jméno programu, poté pošle signál všem procesům daného programu. (Něco jako killall, ale killall není standardní příkaz. Doporučuji podívat se na parametr -o příkazu ps.) (4b)
4. Napište skript, který na vstupu očekává řádky tvaru:
   ```
   cil : z1 z2 .... zn
   ```
   kde cil, z1, z2, ..., zn jsou názvy souborů. Ze vstupu vypíše skript ty cíle, které jsou starší než některý ze souborů z1, ..., zn na témže řádku (závislosti). (Parametr -nt příkazu test není standardní, použijte find s parametrem -newer.) (4b)

14. cvičení (26.5. 2010)

Příklady:

1. Vytvořte skript, který bude generovat obsah pro html, ve vstupním souboru učiní ze všech nadpisů uzavřených mezi tagy <h[1-4]> záložku uzavřenou do <a name="...">. Současně na standardní výstup vypíše skript seznam nadpisů s <a href="#...">, které budou odpovídat příslušným jménům nadpisům v souboru.
2. Napište skript, který čte standardní vstup a vypíše N nejdelších řádků, kde N je zadáno jako parametr na příkazové řádce. Pokud je skript spuštěn bez parametrů, použije se N=10. Skript musí kontrolovat, že má správný počet parametrů (0 nebo 1). Skript by měl řádky vypsat v pořadí v jakém se vyskytují ve vstupu a u každé řádky bude na začátku uvedeno pořadí řádky v rámci vstupu a počet znaků na této řádce.