Domácí příprava k předmětu YD38UOS

Úloha 10 – práce s lokálními soubory – pokračování

- nastudujte si programy **find**, **chmod**, **ls**
- nastudujte si problematiku přidělování práv v unixových systémech (práva pro uživatele, skupinu a ostatní)

10/1 vytvořte libovolný soubor a nastavte práva tak, aby jej nemohl číst nikdo jiný než vy

```
touch ch.txt ; chmod 400 ch.txt
```

nebo

```
touch ch.txt && chmod u=r ch.txt
```

10/2 vytvořte adresář a v tomto adresáři vytvořte několik souborů. Nastavte přístupová práva k adresáři tak, aby nebylo možné zjistit, jaké soubory jsou v adresáři (program **ls** nebude fungovat), ale pokud napíšete správnou cestu k souboru, tak ho budete moci číst (například programem **cat**)

```
mkdir test
cd test
touch s{1..3}.txt
cd ..
chmod 111 test nebo chmod a=x test        NENI ODZKOUSENO !!!
zkouska     ls = vypise error
            cat test/s1.txt = vypise obsah s1
```

10/3 pomocí programu **find** nalezněte v aktuálním adresáři a ve všech podadresářích soubory, které byly změněný během posledních 24 hodin.

```
find -mtime -1 -print
```

10/4 ve vašem domovském adresáři nalezněte všechny soubory, které jsou typu symbolický odkaz.

```
find -type l
```

- nastudujte si program **tar**, **gzip** a **gtar**

10/5 vytvořte si testovací adresář s nějakými soubory a tento adresář archivujte a komprimujte (archivace pomocí programu **tar** a komprese pomocí programu **gzip**). Výsledek rozbalte na jiné místo.

```
mkdir test
cd test
touch s{1..5}.txt
cd ..
tar -cf (nebo cvf?) test.tar test
gzip -1 test.tar           #(jednim prikazem> tar czf test.tar.gz test)
mkdir test2
mv test.tar.gz test2
cd test2
cat test.tar.gz | gzip -d | tar xf -
ls test                    # pokusny vypis po rozbaleni
```

nebo

```
gunzip test tar.gz
tar -xvf test.tar
```

Úloha 11 – práce s procesy

- nastudujte si programy
  ps = vypis procesu
  ptree = zobrazeni stromu procesu
  kill = zasilani signalu
  top = vypis bezicich procesu

pojem **PID** (process ID) = zjistuji se, aktualni a history

11/1 vypište všechny spuštěné procesy na aktuálním počítači

```
ps –ef
```

- pomocí programu **top** si prohlédněte, které aplikace zabírají nejvíce paměti a které nejvíce procesorového času

```
top
```

11/2 vypište PID aktuálního shellu pomocí proměnné prostředí **$$**

```
echo $$
```

11/3 vypište strom procesů aktuálního shellu

```
ptree
```

nebo

```
ptree -c    nebo -a        nebo -z
```

- nastudujte si práci s úlohami – příkazy
  fg = na pozadi
  bg = na popredi
  jobs = vypise aktualne probihajici ulohy

  a řídící znaky
  & =
  CTRL+Z = presun na fg nebo bg

11/4 spusťte na popředí program **sleep 120** a následně jej přesuňte na pozadí (**CTRL+Z** a příkaz **bg**)

```
sleep 120
CTRL+Z bg 1         # 1 = cislo ulohy
```

11/5 spusťte příkaz **sleep 150** na pozadí (znak **&** na konci příkazové řádky)

```
sleep 150 &
```

11/6 vypište si aktuálně spuštěné úlohy na pozadí pomocí příkazu **jobs**

```
jobs
```

11/7 jednu z úloh na pozadí násilně ukončete programem **kill**, pro tento účel potřebujete znát PID úlohy. To zjistíte příkazem **ps** nebo příkazem **jobs -pl**.

```
ps                                              # vrati cislo (PID) ulohy
kill -9 PID nebo kill -SIGKILL PID              # dosadi cislo ulohy z predchoziho prikazu
```

11/8 druhou úlohu přesuňte zpět na popředí příkazem **fg**

```
fg cislo ulohy     nebo PID ???
```

- nastudujte si využití příkazu **time**
- pro testoví využijeme program **dd** na kopírování bloků dat
- příkaz **dd if=/dev/zero of=/dev/null count=64 bs=1k** vygeneruje a zahodí 64kB nulových znaků

```
time dd if=/dev/zero of=/dev/null count=64 bs=1k        #vrati cas spotrebovany ulohou
```

- příkaz **dd if=/dev/urandom of=/dev/null count=64 bs=1k** vygeneruje a zahodí 64kB pseudonáhodných čísel

```
time dd if=/dev/urandom of=/dev/null count=64 bs=1k
```

- příkaz **dd if=/dev/random of=/dev/null count=64 bs=1k** vygeneruje a zahodí 64kB náhodných čísel

```
time dd if=/dev/random of=/dev/null count=64 bs=1k
```

- porovnejte dobu potřebnou pro vykonání jednotlivých příkazů, využijte k tomu příkaz **time**

  CO TIM BASNIK MINIL ??? vizuelne posoudit aneb kdo vidi ten vi

Úloha 12 – standardní vstup, výstup a chybový výstup

- nastudujte si co znamenají pojmy standardní vstup, standardní výstup a standardní chybový výstup
- zjistěte si význam jednotlivých speciálních zařízení **/dev/null**, **/dev/stdin**,

  /dev/stdout, **/dev/stderr**
- nastudujte si význam znaků pro přesměrování **<**, **>**, **>>**, **|**

12/1 spočítejte počet řádků v souboru **/usr/dict/words** s použitím přesměrování standardního vstupu na soubor (znak **<**)

```
wc –l</usr/dict/words
```

12/2 uložte výstup programu **date** do souboru **datum** (znak **>**)

```
date > datum
```

12/3 připište znovu výstup programu **date** do souboru **datum** tak, aby v něm zůstal i původní záznam (znak **>>**)

```
date >> datum
```

12/4 přesměrujte výstup příkazu **ls ~ neexistuje** tak, aby chybové hlášky byly v souboru **ls.err** a výstup programu v souboru **ls.out**. (přesměrování chybového výstupu pomocí **2>**)

```
ls neexistuje >ls.out 2>ls.err
```

12/5 pomocí příkazu **echo** vypište libovolnou hlášku na standardní chybový výstup (přesměrování pomocí **>&2** nebo pomocí **> /dev/stderr**)

```
echo „Pokusny text“ >&2    nebo    echo „Pokusny text“ >/dev/stderr
```

12/6 vytvořte si adresář a nastavte práva tak, aby nikdo nemohl nic (a-rwx)

```
mkdir test && chmod 000 test nebo chmod a-rwx test
```

12/7 spusťte program **find** tak aby vypsal obsah vašeho domovského adresáře

```
find $HOME  nebo   find ~
```

12/8 přesměrujte standardní výstup do programu **wc** – spočítejte kolik souborů je ve vašem domovském adresáři (použijte znak **|**)

```
find $HOME | wc -l         nebo ls ~ | wc -l
```

12/9 přesměrujte chybový výstup tak, aby zavolání předchozího příkazu nevypsalo žádnou chybovou hlášku (použijte přesměrování chyb do **/dev/null**)

```
find $HOME 2> /dev/null
```

Úloha 13 – proměnné prostředí

- zopakujte si příkaz **echo** a nastudujte si příkazy

  env = prikaz – vypise vsechny promenne prostredi/systemu
  set = vypise vsechny promenne systemu
  unset = zrusi promenne
  export = exportuje promennou do jineho (nadrazenoho !!!) sh
  source = nacte script s promennou y nizsiho do vyssiho sh, aby bylo mozno spustit promennou

13/1 nastavte proměnnou **a** na hodnotu **pozdrav**

```
a=pozdrav
echo $a
```

13/2 do proměnné **b** nastavte počet řádku v souboru **/usr/dict/words** – použijte konstrukci **$(prikaz)**

```
b=$(wc -l < /usr/dict/words)      nebo    b=`wc -l < /usr/dict/words`
echo $b
```

13/3 do proměnné **c** nastavte číslo z proměnné **b** vynásobené číslem 2 – použijte konstrukci **$((vyraz))**

```
c=$(($b*2))
echo $c
```

13/4 vypište obsah proměnné **a**, **b** a **c** (příkaz echo a znak **$** před názvem proměnné).

```
echo $a $b $c
```

13/5 uklidťte po sobě všechny proměnné, které jste vytvořili (příkaz **unset**)

```
unset a b c
```

Úloha 14 – program AWK

- nastudujte si program **awk**, význam jednotlivých části příkazu (**BEGIN**, **END**, regulární výrazy, **{}**, **print**, aritmetické operátory a pod.)
- nastudujte si možnosti rozdělení vstupních dat podle nějakého znaku (proměnná **FS**) a následné zpracování jednotlivých částí (**$0**, **$1**, **$2**, **$3**)
- nastudujte si možnost psaní **awk** skriptů do externího souboru a jejich volání pomocí **awk –f soubor**

```
awk /vzor/ {prikaz}
ls -l | cat -n | awk '{print $2,$4,$10}'
ls -l | cat -n | awk '{print $2 " vl: "$4, $10}'
ls -l | cat -n | awk '{print $0}' = ls -l | cat -n
ls -l | cat -n | awk '/^ *1/{print $0}'
```

14/1 vytvořte jednoduchý program pro **awk**, který bude opisovat vše ze standardního vstupu na výstup (použití proměnné **$0**, zadávání provádíte přes klávesnici, konec vstupu je klávesová zkratka **CTR+D**)

```
awk '{print $0}'
# libovolne se vypisuje na obrazovku, program se ukonci CTRL+D
```

14/2 vytvořte jednoduchý program pro **awk**, který v průběhu nic nevypisuje, ale na konci vypíše počet zpracovaných řádků (proměnná **NR**)

```
awk 'END{print NR}'
# NR = number of records = cislo zaznamu/vstupni radky
```

14/3 použijte výstup příkazu **file /usr/bin/*** pro další práci (přesměrovávejte do programu **awk** pomocí **|**)

- napište **awk** skript, který rozdělí výstup podle znaku „:“ a vypíše pouze druhý sloupec

```
file /usr/bin/* | awk -F: '{print $2}'
```

- napište **awk** skript, který vypíše položky z prvního sloupce takové, které mají v druhém sloupci v popisu slovo **ELF** (použijte konstrukci pro testování regulárním výrazem **$2 ~ /ELF/ { prikazy k vykonani; }**)

```
file /usr/bin/* | awk -F: '$2~ /ELF/{print $1}'
```

nebo

```
file /usr/bin/* | awk -F: '/:.*ELF/{print $1}'
```

- napište **awk** skript, který zjistí počet binárních programů a počet skriptů (testování druhého sloupce na obsah textu **ELF** a **script**)

```
file /usr/bin/* | awk -F: '$2~ /ELF/ || $2~ /script/{print $1}'
```

- napište **awk** skript, který v procentech vypíše kolik je ve výpisu binárních programů, kolik skriptů a kolik všeho ostatního (výstup bude vypadat zhruba takto: **binarni: 75.23% skripty: 21.15% ostatni: 3.62%**)

```
touch awk.awk

# file /* awk -F: -f awk.awk      volani skriptu

$2~ /ELF/{Nelf=Nelf+1}
$2~ /script/{Nsc++}
END {print NR, "binarni:" Nelf*100/NR"% scripty:" Nsc*100/NR"% ostatni:" (NR-Nelf-Nsc)*100/NR"%"}
```

14/4 napište **awk** skript, který zpracuje výstup programu **ypcat passwd** nebo souboru **/etc/passwd** tak, aby

- výstupem skriptu byl seznam různých shellů a počet uživatelů, který je používá (Použijte asociativní pole **pocet[$7]+=1**. Shelly jsou v sedmém sloupci. V části **END** vypisujte pole pomocí cyklu **for(polozka in pocet)**)

```
# ypcat passwd | awk -F: -f
{pocet[$7]++}
END{
   for (s in pocet) {
      print "shell: " s " pouziva " pocet [s] " uzivatelu"
   }
}
```

- upravte skript tak, aby počítal pouze členy skupiny 1001 (čtvrtý sloupec)

```
# ypcat passwd | awk -F:
$4~ /1001/{pocet[$7]++}
END{
   for (s in pocet) {
      print "shell: " s " pouziva " pocet [s] " uzivatelu"
   }
}
```

14/5 v textovém souboru máte napsány řádky s čísly, vytvořte **awk** skript, který je sečte a vypíše výsledek. Pro následující příklad souboru vyjde součet 21.

jedna
dve
pet
pet
dve
pet
jedna

```
# cat cisla | awk -f cisla.awk    nebo    awk -f cisla.awk cisla
$4~ /1001/{pocet[$7]++}
END{
   for (s in pocet) {
      print "shell: " s " pouziva " pocet [s] " uzivatelu"
   }
}
```

Úloha 15 – vytváření vlastních skriptů pro bash

- zopakujte si příkazy **source**, **export**, práci s textovým editorem a komentování kódu pomocí znaku #.

15/1 vytvořte si jednoduchý skript s názvem **s1**, který vypisuje obsah proměnné **a** a nastavuje proměnnou **b** na libovolnou hodnotu (skript musí mít na prvním řádku uvedeno **#!/bin/bash** a musí být spustitelný – **chmod u+x s1**)

```
touch s1 && chmod u=700 s1  nebo    touch s1 ; chmod u=700 s1
```

```
#!/bin/bash
echo $a
export b=blablabla

# spusteni ./s1
```

15/2 v aktuálním shellu nastavte proměnnou **a** na libovolnou hodnotu a spusťte předchozí skript (zadejte příkaz **./s1**) a pozorujte co se vypíše

```
a=aaa              # nevypise se nic, protoze neni export
```

15/3 v aktuálním shellu vytvořte z lokální proměnné **a** globální proměnnou (příkaz **export**), znovu spusťte skript **s1** a pozorujte co se vypíše

```
export a=aaa       # pokud se nevyexportuje, pak si ji skript neslizne
```

15/4 v aktuálním shellu vypište obsah proměnné **b**, zjistíte, že proměnná není nastavena, i přesto, že je ve skriptu **s1** nastavována na nějakou hodnotu

```
echo $b
```

15/5 upravte skript **s1** tak, aby exportoval proměnnou **b** a zkuste ji znovu vypsat v aktuálním shellu

```
export b=blablabla         # echo nevypise NIC, protoze promenna $b je v nizsim sh
```

15/6 spusťte skript **s1** příkazem **source s1** a zkuste znovu vypsat obsah proměnné **b**

```
source s1                  # zavola promennou z nizsiho sh
echo $b                    # vypise promennou $b
```

15/7 vysvětlete předchozí chování a shrňte, za jakých podmínek je obsah proměnné vidět ve spouštěném skriptu a za jakých podmínek může skript nastavit proměnné prostředí v aktuálním shellu

```
export - z vyssiho sh do nizsiho

z nizsiho do vyssiho nutno        export
                                  zavolat source
                                  vypsat
```

15/8 celý skript okomentujte (použijte znak #)

Úloha 16 – návratová hodnota programu

- každý program při svém ukončení vrací číslo, které je uloženo do proměnné **$?**. Pokud je číslo nulové, program proběhl úspěšně
- nastudujte si logické operátory v příkazové řádce

  && = pokud se provede *prikaz1*, pak se provede i *prikaz2*
  || = pokud se neprovede *prikaz1*, pak se provede *prikaz2*

  ; = provede se bez ohledu na to, zda se provedl *prikaz1*

16/1 vypište obsah aktuálního adresáře (program **ls**) a následně vypište návratovou hodnotu

```
ls && echo $?       nebo    ls ; echo $?
```

16/2 vypište obsah neexistujícího adresáře a vypište návratovou hodnotu

```
ls homee || echo $?         nebo    ls homee ; echo $?
```

16/3 vypište obsah aktuálního adresáře a výstup filtrujte pomocí programu **grep**

```
ls | grep [.]*      # vvypise vse
ls | grep et*       # vypise vse co obsahuje 'et'
                    atd.
```

- filtrujte regulárním výrazem, kterým projdou nějaké soubory a vypište návratovou hodnotu

```
ls | grep '^[m].*' && echo $?     # pokud existuji soubory '^[m]', pak vypise 0
```

- filtrujte výrazem, kterým neprojdou žádné soubory a vypište návratovou hodnotu

```
ls | grep '^[z].*' || echo $?     # pokud neexistuji soubory '^[z]', pak vypise 1
```

16/4 nastudujte si příkaz **test**

- otestujte, jestli je adresář **/usr/bin** symbolický odkaz (vypište návratovou hodnotu)

```
test -L /usr/bin ; echo $?
```

- otestujte, jestli je adresář **/bin** symbolický odkaz (vypište návratovou hodnotu)

```
test -L /bin ; echo $?
```

- do proměnných **a** a **b** si uložte nějaká čísla, otestujte jestli jsou čísla shodná

```
a=1
b=2
test $a -eq $b ; echo $?
```

- otestujte, jestli je **s1** soubor, pokud ano, vypište text „je soubor“ (použijte operátor **&&**)

```
test -f s1 && echo "je soubor"
```

- zjistěte, jestli existuje adresář **adr1**, pokud ne, vytvořte jej (použijte operátor **||**)

```
test -d adr1 || mkdir adr1
test -d adr1 || mkdir adr1 && ls  # s kontrolnim vypisem
```

Úloha 17 – parametry v příkazové řádce

17/1 nastudujte si význam proměnných

$# = cislo posledniho argumentu**, počet argumentu**
$0 = nazev scriptu
$1, **$2**, … = promenna

a význam příkazu **shift** = pri cyklu posune o jednu promennou vpravo a cyklus zopakuje

17/2 napište jednoduchý skript pro bash, který vypíše součet dvou čísel. Čísla budou předávána jako parametry na příkazové řádce. Příklad spuštění skriptu: **./soucet 1 5** sečte čísla 1 a 5.

```
touch soucet

#!/bin/bash

if [ $# -eq 2 ]
   then
      echo $(($1+$2))
   else
      echo "chyba - nevis kolik je DVA?"
fi
```

17/3 napište jednoduchý skript, který bude archivovat adresář zadaný v příkazové řádce. Výsledný zkomprimovaný archiv bude mít stejné jméno jako adresář, navíc s příponou **.tar.gz**. Příklad spuštění skriptu: **./komprese adresar1** vytvoří zabalený soubor **adresar1.tar.gz**.

```
# predpoklad – existuje realny (a plny) adresar1

touch komprese

#!/bin/bash

if [ -d $1 ]
   then
      tar czf $1.tar.gz $1
   else
      echo "tak mas smulu"
fi
```

Úloha 18 – podmínky a cykly

- nastudujte si význam klíčových slov **for**, **while**, **do**, **done**, **if**, **then**, **fi**, **case**, **esac**.

18/1 napište skript, který vypíše všechny parametry na příkazové řádce pod sebe. Využijte cyklus **while**, počítání parametrů na příkazové řádce pomocí **$#** a příkaz **shift** pro odebrání parametru.

```
#!/bin/bash

while ((#>0))
   do
      echo $1
      shift
   done
```

18/2 napište skript, který sečte všechna čísla předaná na příkazové řádce (obdoba předchozího zadání).

```
#!/bin/bash # soucet vsechn parametru var1

SUM=0
while ((#>0))
   do
      echo $((SUM=SUM+1))
      shift
   done
      echo "vysledek = $SUM"


#!/bin/bash # soucet vsechn parametru var2

SUM=0
while ((#>0))
   do
      ((SUM=SUM+1))
      shift
   done
      echo "vysledek = $SUM"
```

18/3 napište skript, který bude odpočítávat dobu na psaní testu. **./test 45**

- skript bude mít jeden parametr a to dobu testu v minutách.
- výstupem skriptu bude nejprve na prvním řádku aktuální čas (čas začátku).
- na dalším řádku bude čas konce (čas začátku zvětšený o délku testu).
- po uplynutí času potřebného na test začne skript vypisovat na obrazovku text „konec“ v nekonečné smyčce. (pořád dokola, dokud uživatel nestiskne CTRL+C)
  - použijte například program **date** na zjištění aktuálního času a program **sleep** na čekání během testu.

```
POZOR !!! Funguje jen v Linuxu, nefunguje v Solarisu ani v putty !!!

#!/bin/bash

if [ $1 -le 0 ]
   then
      echo "chybne zadani"
   else
      echo "aktualni cas: `date '+%H:%M:%S'`"
      echo "cas konce: `date -d "+$1min" '+%H:%M:%S'`"
      sleep $(($1*2))
   until [ 1 -eq 2 ]
   do
      echo "konec"
   done
fi
```

18/4 napište jednoduchou kalkulačku **./kalkulacka 3 p 10**

- skript bude mít tři parametry. První a třetí parametr bude číslo, druhý parametru bude písmeno. Pokud bude počet parametrů jiný než 3, skript se ukončí s nahlášením chyby.
- podle druhého parametru se bude provádět příslušná operace (d-děleno, k-krát, p-plus, m-minus)
- skript si řádně okomentujte
- využijte příkaz **case**/**esac**.

```
touch kalkulacka

#!/bin/bash

if (($#!=3))
  then
      echo "usage $0 p1 p2 p3"


            # $0 = jmeno souboru (melo by odpovidat jmenu skriptu)
                    exit 1
            # kazdy program po skonceni za sebou necha navratovou hodnotu
            # exit = prikaz pro navratovou hodnot $?
            # exit = 1..=not true, 0=true


   else
      case $2 in
         p) echo $((SUM=$1+$3));;
         m) echo $((SUM=$1-$3));;
         k) echo $((SUM=$1*$3));;
         d) echo $((SUM=$1/$3));;
         *) echo "error operation :$2"; exit 2;;
   esac
fi
```