Úvod do UNIXu

Přehled základních příkazů

- **man** – zobrazí manuálovou stránku příkazu
- **ls** – vypíše obsah adresáře *(list segments)*
  - `ls -a` ... vypíše všechny soubory včetně skrytých
  - `ls -d .*` ... vypíše všechny skryté soubory (-d ... expanze)
  - `ls -lS` ... vypíše všechny soubory seřazené podle velikosti
  - `ls -lt` ...vypíše všechny soubory seřazené podle času (nejnovější je na začátku seznamu)
- **cd** – změní pracovní adresář *(change directory)*
  - `cd -` ... o krok zpátky
- **pwd** – vypíše pracovní adresář *(print working directory)*
- **mkdir** – vytvoří nový adresář *(make directory)*
- **rmdir** – odstraní prázdný adresář *(remove directory)*
- **mv** – přesune soubory *(move)*
- **cp** – zkopíruje soubory *(copy)*
- **rm** – smaže soubory *(remove)*
- **wc** – spočítá řádky, slova a bajty v souboru *(word count)*
  - `wc -l soubor` ... spočítá počet řádek v souboru
- **cat** – zřetězí a vypíše obsah souborů *(concatenate)*
- **head** – vypíše prvních 10 řádek souboru
  - `head -42` ... vypíše všechny řádky do prvních 42
- **tail** – vypíše posledních 10 řádek souboru
  - `tail -n+2` ... vypíše všechny řádky bez prvních dvou (ze vstupu)
- **echo** – opíše své parametry na standardní výstup
- **date** – vypíše aktuální čas a datum ve zvoleném formátu
- **touch** – změní čas posledního přístupu k souboru; v případě neexistence vytvoří nový soubor
  - `touch -c` ... jako normální touch, ale nevytvoří nový soubor
  - `touch -m` ... změní modification time na aktuální
  - `touch -t [[CC]YY]MMDDhhmm[.SS]` ... specifikuje čas modifikace souboru
    - CC – Specifies the first two digits of the year
    - YY – Specifies the last two digits of the year. If the value of the YY is between 70 and 99, the value of the CC digits is assumed to be 19. If the value of the YY is between 00 and 37, the value of the CC digits is assumed to be 20. It is not possible to set the date beyond January 18, 2038.
    - MM – Specifies the month
    - DD – Specifies the date
    - hh – Specifies the hour
    - mm – Specifies the minute
    - SS – Specifies the seconds

Nastavení modification time souboru na 11.5.2001, pokud neexistuje, tak ho nezaložit:

```
touch -c -m -t 200105111155 soubor
```

`.` = aktuální adresář

`..` = nadřazený (parent directory)

???

`/bin/sh ...` *shell
`scp ...` vzdálené kopírování (nefunguje na putty)
`last ...` historie nalogovaných uživatelů
`tee`

`ps ...` aktuálně běžící procesy
`df ...` vypíše informace o využití disku
`du ...` info o využití disku aktuálního adresáře

More & Less

Pro dlouhé výstupy:

- *prikaz* more
- *prikaz* less

Stat

`stat` *soubor* `...` vypíše informace o souboru
Výstup:

```
  File: 'a.txt'
  Size: 6               Blocks: 2          IO Block: 4096   regular file
Device: 1ah/26d    Inode: 466616594   Links: 1
Access: (0644/-rw-r--r--)  Uid: (63658446/mohwalda)   Gid: (  200/students)
Access: 2001-05-11 11:55:00.000000000 +0200
Modify: 2001-05-11 11:55:00.000000001 +0200
Change: 2001-05-11 11:55:00.000000000 +0200
Birth: -
```

N-tý řádek ze vstupu

`sed -n 'Np'` ... vrátí N-tý řádek ze vstupu

Výpis 24. až 42. řádky ze vstupu (z příkazu seq):

```
seq 100 | head -42 | tail -n+24
```

Maily

`mail prijemce ...` odešle text ze vstupu přijemci

`echo "Text emailu..." | mutt -a "priloha" -s "Subject" -- email-prijemce ...` email s přílohou

Práva a spouštění souborů

chmod

`...` nastavuje práva pro vlastní soubory (RWX)

- Formát:
  - `chmod | _ _ _ | _ _ _ | _ _ _ soubor`
  - `chmod _ _ _ soubor`
    - hodnota & právo> 0 --- 1 --x 2 -w- 3 -wx 4 r-- 5 r-x 6 rw- 7 rwx
  - \+ přidává, - ubírá, = nastavuje

Odberání read permissionu pro others adresáře *directory* a jeho podadresářů:

```
chmod -R o-r directory
```

Set a

- **read/write/execute permissions for owner**
- **read/execute for group**
- **read-only for other to a file**

, via octal numeric way and via symbolic way.

```
chmod 754 file
chmod u=rwx,g=rx,o=r file
```

umask

`...` nastavuje omezení pro defaultní práva nově vytvořených souborů (tzn. negace práv z chmod pro nově vytvořené soubory)

- pro adresáře je defaultně 777
- pro soubory je defaultně 622 ???

chown

`...` nastavuje vlastníka

`sh file.sh ...` spustí script

Nahrazování znaků

`tr "stary znak" "novy znak" ...` nahradí staré znaky novými
`tr "abc" "xyz"`
`tr "[0-9]" "[a-h]"`
`tr -s "znak" ...` maže z více znaků pouze první (za hodně znaků nahradí jeden)
`tr -s "[:alnum:]" "-" > /tmp/gid ...` nahradí jakýkoli souvislý string číslic nebo znaků za "-"
`tr -d "znak" ...` maže každý znak

Nahrazení znaků v souboru (a -> A):

```
cat soubor | tr "a" "A" > temp
mv temp soubor
```

Znakové třídy:

```
[:alnum:]       all letters and digits
[:alpha:]       all letters
[:blank:]       all horizontal whitespace
[:cntrl:]       all control characters
[:digit:]       all digits
[:graph:]       all printable characters, not including space
[:lower:]       all lower case letters
[:print:]       all printable characters, including space
[:punct:]       all punctuation characters
[:space:]       all horizontal or vertical whitespace
[:upper:]       all upper case letters
[:xdigit:]      all hexadecimal digits
[=CHAR=]        all characters which are equivalent to CHAR
```

Wildcardové patterny:

*	Match zero or more characters.
?	Match exactly one occurrence of any character.
|	An or expression. The substrings used with this operator can contain other special characters such as * or $. The substrings must be enclosed in parentheses, for example, (a|b|c), but the parentheses cannot be nested.
$	Match the end of the string. This is useful in or expressions.
[abc]	Match one occurrence of the characters a, b, or c. Within these expressions, the only character that needs to be treated as a special character is]; all others are not special.
[a-z]	Match one occurrence of a character between a and z.
[^az]	Match any character except a or z.
*~	This expression, followed by another expression, removes any pattern matching the second expression.

Práce s tabulkami cut & paste

`cut -d";" -f1,3 soubor.in` ... vypíše 1. a 3. sloupec v .csv souboru (sloupce **oddělený ;**)
`paste -d: _ _ _ _` ... vloží soubory do tabulky (každý soubor 1 sloupec **oddělený :**)

Změna malých písmenek na velká v 1. sloupci:

```
cut -d";" -f1 calories.csv | tr "[:lower:]" "[:upper:]" > prvni
cut -d";" -f2- calories.csv > zbytek
paste -d";" prvni zbytek > calories2.csv
```

Přepsání GID (3. sloupce) v /etc/group:

```
cut -d":" -f1,2 /etc/group > /tmp/prvni
cut -d":" -f3 /etc/group | tr -s "[:alnum:]" "-" > /tmp/gid
cut -d":" -f4- /etc/group > /tmp/posledni
paste -d: /tmp/prvni /tmp/gid /tmp/posledni > group2
```

You have a file corpus-small.txt containing English sentences and their translation, each on separate line, each sentence pair separated by an empty line. Transform the file, so that on each line it contains both the English sentence and its translation, separated by [TAB]. (The file must contain only two columns.)

$ cat corpus-small.txt

$10,000 Gold?
Zlato za 10Â 000 dolarĹŻ?

SAN FRANCISCO â€“ It has never been easy to have a rational conversation about the value of gold.
SAN FRANCISCO â€“ VĂ©st racionĂˇlnĂ rozhovor o hodnotĂ› zlata nikdy nebylo snadnĂ©.

```
sed -n '1~3p' corpus-small.txt > /tmp/corpus-eng
sed -n '2~3p' corpus-small.txt > /tmp/corpus-cze
paste /tmp/corpus-eng /tmp/corpus-cze > corpus-small1.txt
```

Nebo:

```
cat corpus-small.txt | tr -s "\n" "\n" > /tmp/corpus
cat /tmp/corpus | paste - - > corpus-small2.txt
```

Sort

`sort` *`soubor`* `...` seřadí soubor podle 1. znaku
`sort -n ...` seřadí podle čísla (řadí podle ASCII)
`sort -k` *`n`* `...` seřadí podle sloupce *n*
`sort -k m,`*`n`* `...` seřadí podle sloupce *m-n*
`sort -r ...` seřadí opačně
`sort -f`
`sort -u ...` seřadí a navíc odstraní identické výskyty za sebou

Uniq

`sort | uniq ...` seřadí řádky a odstraní duplicitní řádky (zbyde jen jedna z duplicitních)
-c na začátek řádky vypíše počet
-d vypíše jen duplicitní řádky

Pozn.

- Při česky psaných slovech je užitečné využít locale -a.
- Proti nesortění hlavičky (1. řádek) využít head ... > head & tail ... > rest, nasortit a pak concatenovat

Count the number of different login-shells used by users listed in "/etc/passwd":

```
cut -d":" -f7 /etc/passwd | sort | uniq -c | wc -l
```

Seřazení /etc/group sestupně podle počtu uživatelů (4. sloupec):

(Hint: extract the column with the users, replace each user by "@" and then sort the file using this modified column. Remove this extra column after sorting the file.)

```
cat /etc/group > /tmp/grsort_4c
cut -d":" -f4 /etc/group | sed -e "s/[a-zA-Z0-9]/1/g" | tr -d "," > /tmp/grsort_extc
paste -d":" /tmp/grsort_4c /tmp/grsort_extc | sort -t":" -k 5,5 -n -r > /tmp/group_sorted
cut -d":" -f1,2,3,4 /tmp/group_sorted > group_sorted
cat group_sorted
```

Vyndavání specifických řádek ze souboru

`sed -n '2p'` ... vrací 2. řádku

`sed -n '1~3p' corpus-small.txt` ... vrací 1., 4., 7., 10., ... řádku ze souboru corpus-small.txt

- -n suppress automatic printing of pattern space

Vyndavání specifických sloupců ze souboru

`cut -d";" -f1,3 soubor.in` ... vypíše 1. a 3. sloupec v .csv souboru (sloupce oddělený ;)

JOIN

`join` *`file1 file2`* `...` spojí soubory

`-i` (ignoruje velikosti písmen)

`-t CHAR` (CHAR - oddělovač prvků ⇒ spojí na základě ekvivalentních hodnot 1. souboru a 2. souboru, kde jsou pouze řádky shodné s CHAR)

`-j` (určuje sloupce např. -j1 -j2 -> spojí první sloupec prvního a druhý sloupec druhého)

`-e EMPTY` (přepíše prázdný tím, co je v EMPTY)

`-a` (??? přidá číslo souboru)

`-1 FIELD` (join on this FIELD of file 1)

`-2 FIELD` (join on this FIELD of file 2) ⇒ klíče

Chceme spojit cz a en země ze 2 souborů:

```
u2-17:~/WORKDIR$ head countrycodes_en.csv
English short name (upper/lower case);Alpha-2 code;Alpha-3
code;Numeric code;ISO 3166-2 codes
"Afghanistan";AF;AFG;004;ISO 3166-2:AF
"Åland Islands";AX;ALA;248;ISO 3166-2:AX
"Albania";AL;ALB;008;ISO 3166-2:AL
"Algeria";DZ;DZA;012;ISO 3166-2:DZ
"American Samoa";AS;ASM;016;ISO 3166-2:AS
"Andorra";AD;AND;020;ISO 3166-2:AD
"Angola";AO;AGO;024;ISO 3166-2:AO
"Anguilla";AI;AIA;660;ISO 3166-2:AI
"Antarctica";AQ;ATA;010;ISO 3166-2:AQ
```

```
u2-17:~/WORKDIR$ head kodyzemi_cz.csv
alpha-3;numeric;alpha-2;"Stát"
AFG;004;AF;"Afghánistán"
ALA;248;AX;"Ålandy"
ALB;008;AL;"Albánie"
DZA;012;DZ;"Alžírsko"
ASM;016;AS;"Americká Samoa"
VIR;850;VI;"Americké Panenské ostrovy"
AND;020;AD;"Andorra"
AGO;024;AO;"Angola"
AIA;660;AI;"Anguilla"
```

```
#!/bin/sh
cut -d";" -f1,4 countrycodes_en.csv | sort -t";" -k2,2 > file1
cut -d";" -f2,4 kodyzemi_cz.csv | sort -t";" -k1,1 > file2
join -t";" -1 2 -2 1 file1 file2 | head
```

```
u2-17:~/WORKDIR$ sh join.sh
004;"Afghanistan";"Afghánistán"
008;"Albania";"Albánie"
010;"Antarctica";"Antarktida"
012;"Algeria";"Alžírsko"
016;"American Samoa";"Americká Samoa"
020;"Andorra";"Andorra"
024;"Angola";"Angola"
028;"Antigua and Barbuda";"Antigua a Barbuda"
031;"Azerbaijan";"Ázerbájdžán"
032;"Argentina";"Argentina"
```

Split

- Rozdělí soubory do dalších souborů.
- **split** outputs fixed-size pieces of input *INPUT* to files named *PREFIX***aa**, *PREFIX***ab**, ...
- The default size for each split file is 1000 lines, and default PREFIX is "**x**". With no INPUT, or when INPUT is a dash ("**-**"), read from standard input.

Přepínače:

- -d pojemnuje suffixy číslicemi, namísto písmen
- -l specifikuje, po kolika řádcích rozděluje
- -n generuje CHUNKy - rozdělí da zadaného počtu
- -b specifikuje velikost výstupního souboru
- -a přenastaví délku suffixů (2 defaultně)

Split the file newfile.txt into three separate files called newaa, newab and newac..., with each file containing 22 bytes of data.

```
split -b 22 newfile.txt new
```

Split the file newfile.txt into files beginning with the name new, each containing 300 lines of text.

```
split -l 300 file.txt new
```

Diff

- Zanalyzuje 2 soubory a vrátí odlišné řádky

Příklad:

```
diff file1.txt file2.txt
```

If **file1.txt** contains the following four lines of text:
I need to buy apples.
I need to run the laundry.
I need to wash the dog.
I need to get the car detailed.
...and **file2.txt** contains these four lines:
I need to buy apples.
I need to do the laundry.
I need to wash the car.
I need to get the dog detailed.

...and the output will be:

```
2,4c2,4
< I need to run the laundry.
< I need to wash the dog.
< I need to get the car detailed.
---
> I need to do the laundry.
> I need to wash the car.
> I need to get the dog detailed.
```

DD

⇒ převod a kopírování souborů

Příklad na zabírání místa na disku:

```
dd if=/dev/zero of test count=1 b>10MB
```

Pozn.
`/dev/null` ... vrací EOF
`/dev/zero` ... vrací \0

Find

```
find . -name '*.txt'     ... soubory s příponou .txt
find . -user root        ... soubory jejichž vlastníkem je root
find . -type f           ... obyčejné soubory
find . -type d           ... adresáře
find . -size +10000c     ... soubory větší než 10000 bajtů
find . -newer file       ... soubory novější než soubor file
find . -mtime -7         ... soubory změněné v posledních 7 dnech

find . -prune            ... hledá jenom zadané soubory, nerozbaluje adresáře
find . -depth            ... zanořuje se do podadresářů (defaultní chování)
```

Užitečné nePOSIXové parametry

```
find . -maxdepth n     ... soubory v hloubce nejvýše n
find . -mindepth n     ... soubory v hloubce alespoň n
find . -size +10M      ... soubory větší než 10 megabajtů
find . -delete         ... smaž všechny odpovídající soubory
```

Mezi podmínkami:

-a (and) (defaultní)
-o (or)
-not

Výpis obyčejných souborů s příponou .txt nebo .sh, které byly buď změněny během posledních 14 dnů nebo jsou větší než 1kB:

```
find . -type f \( -name '*.txt' -o -name '*.sh' \) \( -mtime -14 -o -size +1024c \)
```

Prohledá a vypíše všechno z / (tzn. celý počítač), obsahující bin

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

In the subtree of /usr/include find the files with which are in depth at least 2 and at most 3 in the subtree:

```
find /usr/include -mindepth 2 -maxdepth 3 -type f
```

In the subtree of directory /etc find all files which are older than /etc/passwd:

```
find /etc ! -newer /etc/passwd
```

List the long information about the files in the /etc directory subtree that belong to the root group and are executable only by the owner and the group.

```
find /bin \( -not -group root \) -perm -ug=x \( -not -perm -o=x \) -exec ls -l {} \;
```

Výpis všech souborů z roota, které nevalstní root:

```
find / -maxdepth 1 ! -user root -executable -type f
```

Smazání všech simlinků z /tmp:

```
find /tmp -type l -delete
```

Parametr -exec

Teď teprve přichází ta pravá magie. find umí soubory nejenom vyhledávat, ale také pro ně provádět nejrůznější akce – přejmenovávat, mazat, vypisovat začátky, … Za parametr -exec můžeme napsat jeden shellovský příkaz. Dvojice znaků {} se nahradí jménem zpracovávaného souboru (i s cestou) a příkaz musí být ukončen sekvencí \;. Opět proto, že znak ; má v shellu speciální význam. Příklad, který vypíše první řádek ze všech nalezených souborů:

```
find . -type f -name '*.sh' -mtime -14 -exec head -n1 {} \;
```

Pokud chceme pro daný soubor provést více příkazů, máme dvě možnosti. První je použít více parametrů -exec napsaných za sebou. Druhou je zavolání příkazu sh -c '*příkazy*', což je jeden příkaz, který spustí shell a zavolá v něm všechny zadané příkazy. Z hlediska findu se ale tváří jako jeden příkaz a proto ho můžeme předat parametru -exec

Xargs

-n určí # argumentů na na příkaz
-L určí # řádků na příkaz
-a file načítá ze souboru
-d delimiter
-0 namísto mezerou ukončuje nullem

Příklady:

```
u2-12:~/WORKDIR$ echo a b c > file
u2-12:~/WORKDIR$ echo d e f >> file
u2-12:~/WORKDIR$ echo g h i >> file
u2-12:~/WORKDIR$ cat file | xargs
a b c d e f g h i
u2-12:~/WORKDIR$ cat file | xargs echo
a b c d e f g h i
u2-12:~/WORKDIR$ cat file
a b c
d e f
g h i
u2-12:~/WORKDIR$ cat file | xargs echo -
- a b c d e f g h i
u2-12:~/WORKDIR$ cat file | xargs -n2  echo
a b
c d
e f
g h
i
u2-12:~/WORKDIR$ cat file | xargs -I{}  echo "-{}-"
-a b c-
-d e f-
-g h i-
u2-12:~/WORKDIR$ cat file | xargs -n 1 | xargs -I{} echo "-{}-" | xargs -n3
-a- -b- -c-
-d- -e- -f-
-g- -h- -i-
u2-12:~/WORKDIR$ cat file | xargs -n 1 | xargs -I{} echo "-{}-"
-a-
-b-
-c-
-d-
-e-
-f-
-g-
-h-
-i-
```

Split the file /etc/passwd into files, each containing one line. Connect the files together in the reverse order. Compare the resulting file with output of either "tac /etc/passwd" or "tail -r /etc/passwd"

```
u2-12:~/WORKDIR$ split -l1 file tmp
u2-12:~/WORKDIR$ ls -r tmp*
tmpac  tmpab  tmpaa
u2-12:~/WORKDIR$ ls -r tmp* | xargs -I{} cat {} >> out
u2-12:~/WORKDIR$ cat out
g h i
d e f
a b c
u2-12:~/WORKDIR$ cat file
a b c
d e f
g h i
```

Nalezení všech souborů končící .txt v aktuálním adresáři a jeho podstromech a vytvoření jeho kopie končící .txt.bak

```
find -name “*.txt” -type f | xargs -I{} cp {} {}.bak
```

Grep

⇒ hledá vzor v souboru nebo vstupu

Varianty:

-G PATTERN is a basic regular expression (BRE) ⇒ asi default

-E PATTERN is an extended regular expression (ERE)

-F PATTERN is a set of newline-separated strings

Další options:

-c vrací počet řádků odpovídající vzoru

-l vrací pouze názvy souborů odpovídající vzoru

Pozor! Například:

```
u-pl23:~/WORKDIR$ find -type f | xargs file | grep -l 'POSIX shell script'
(standard input)
```

-q nevrací na vstup

-n vrací čísla řádků odpovídající vzoru

-i ignore case distincitons

-v inverz

-o nevrací řádek, ale string, jenž matchuje

From calories.csv, select all lines where food name contains a number (first column).

```
grep -o '^[^;]*[0-9][^;]*;' calories.csv
```

From calories.csv, select all lines where the unit of measure is either "cup" or "oz" (second column).

```
grep '^[^;]*;[[:digit:]\.*]* \(Cup\|oz\)' calories.csv
```

Print all files from /usr/lib that contain 5-7 characters in their name not counting the suffix (everything after the first fullstop).

```
ls /usr/lib | grep '[^\.]\{5,7\}\.'
```

Print all files from /usr/include that do not contain "stdio" string inside of them.

```
find /usr/include/* -type f | xargs -I{} grep 'stdio' {}
```

Výpis všech souborů v /usr/lib a v jeho podstromě, začínající lib a nekončící příponou s číslicemi.

```
find /usr/lib/lib* -type f | grep -v '.*\.[0-9][0-9]*$'
```

Počítání počtu nějakého znaku na řádku:

```
echo "text,text,text,text" | grep -o "," | wc -l
```

Výpis souborů, které jsou shell-scripty:

```
find -maxdepth 1 -type f | xargs file | grep -h 'POSIX shell script' | sed 's/\: *POSIX shell script.*$//p'
```

http://www.grymoire.com/Unix/Regular.html

.	Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor-, character-encoding-, and platform-specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, a.c matches "abc", etc., but [a.c] matches only "a", ".", or "c".
[ ]	A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches "a", "b", or "c". [a-z] specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: [abcx-z] matches "a", "b", "c", "x", "y", or "z", as does [a-cx-z]. The - character is treated as a literal character if it is the last or the first (after the ^, if present) character within the brackets: [abc-], [-abc]. Note that backslash escapes are not allowed. The] character can be included in a bracket expression if it is the first (after the ^) character: []abc].
[^]	Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than "a", "b", or "c". [^a-z] matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.
^	Matches the starting position within the string. In line-based tools, it matches the starting position of any line.
$	Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.

( )	Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, *\n*). A marked subexpression is also called a block or capturing group. **BRE mode requires \(\).**
n	Matches what the *n*th marked subexpression matched, where *n* is a digit from 1 to 9. This construct is vaguely defined in the POSIX.2 standard. Some tools allow referencing more than nine capturing groups.
*****	Matches the preceding element zero or more times. For example, ab*c matches "ac", "abc", "abbbc", etc. [xyz]* matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. (ab)* matches "", "ab", "abab", "ababab", and so on.
{*m*,*n*}	Matches the preceding element at least *m* and not more than *n* times. For example, a{3,5} matches only "aaa", "aaaa", and "aaaaa". This is not found in a few older instances of regexes. **BRE mode requires \{*m*,*n*\}.**

Examples:

- .at matches any three-character string ending with "at", including "hat", "cat", and "bat".
- [hc]at matches "hat" and "cat".
- [^b]at matches all strings matched by .at except "bat".
- [^hc]at matches all strings matched by .at other than "hat" and "cat".
- ^[hc]at matches "hat" and "cat", but only at the beginning of the string or line.
- [hc]at$ matches "hat" and "cat", but only at the end of the string or line.
- \[.\] matches any single character surrounded by "[" and "]" since the brackets are escaped, for example: "[a]" and "[b]".
- s.* matches s followed by zero or more characters, for example: "s" and "saw" and "seed".

Shrnutí regexů (s rozšířením po POSIXu):

abc…	*Letters*
123…	*Digits*
\d	*Any Digit*
\D	*Any Non-digit character*
.	*Any Character*
\.	*Period*
[abc]	*Only a, b, or c*
[^abc]	*Not a, b, nor c*
[a-z]	*Characters a to z*
[0-9]	*Numbers 0 to 9*
\w	*Any Alphanumeric character*
\W	*Any Non-alphanumeric character*
{m}	*m Repetitions*

`{m,n}`	*m to n Repetitions*
`*`	*Zero or more repetitions*
`+`	*One or more repetitions*
`?`	*Optional character*
`\s`	*Any Whitespace*
`\S`	*Any Non-whitespace character*
`^…$`	*Starts and ends*
`(…)`	*Capture Group*
`(a(bc))`	*Capture Sub-group*
`(.*)`	*Capture all*
`(abc|def)`	*Matches abc or def*

Sed

http://www.abclinuxu.cz/clanky/navody/unixove-nastroje-9-sed-nahrazovani-textu

Přepínače:
-e file přidá skriptovací soubor
-n (silent)

Substituce:
`sed 's/pattern/replacement/g'` nahradí každý pattern za replacement -> prochází znak po znaku
s# ... # ... # matchování toho za ...

Výpis řádek:
`sed -n '/row/p'` vypíše zadané řádky plodle row

Zakomentování všech řádek v /etc/passwd, kde GID = UID (3. sloupec = 4. sloupec):

```
cat /etc/passwd | sed 's/^[^:]*:[^:]*\(:[[:digit:]]*\)\1/# &/g'
```

Sed script, který prohodí pořadí znaků na každém řádku:

```
#!/bin/sh$
sed '
s/^/~/
:label
  s/\(~.*\)\(.\)/\2\1/
  t label
s/~//
'
```

File

Vrací typ souboru tak. že do něj nakoukne a vypíše nějaké basic info.

```
$ file soubor.c
 soubor.c: C program text
```

SHELL

basic manuál: https://linuxconfig.org/bash-scripting-tutorial#h2-simple-backup-bash-shell-script
Viz výše...

1. Write a script in sed which performs formatting of comments in the shell script in a following way:
   a. If there isn't a "#!.*" on a first line, insert "#!/bin/sh" there.
   b. If there is a comment with trainling whitespaces in front of it (e.g. "# <something>", delete the whitespaces
   c. If there is some text in front of a comment, print a comment first on a separate line and on the second line, print the text which was in front of the comment.
   d. If there is an empty comment ("#" that is not followed by any text), delete it.

```
$ cat > format.sh <<END
#!/bin/sed -f

# We need to apply the changes in the following order.
# a.
1{
s/^\([^#][^!].*\)$/#!\/bin\/sh\n\1/
}

# d.
s/#[^[:alnum:]]*$//

# b.
s/^ *#/#/

# c.
s/^\(.*[^ ].*\)\(#.*$\)/\2\n\1/
END
```

2. Write a script, that takes n+1 parameters, first parameter contains a number x between 1 and n. The scripts then outputs the value of the (x+1)-th parameter.

```
$ cat > params.sh <<END
#!/bin/sh

shift \$1
echo \$1

END
```

3. What will the following lines print to the screen?

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

```
hello
hi
hi
---
hello
hi
hello
---
hi
hello
---

hi
---

---
ahoj
```

```
u-pl0:~/WORKDIR$ a=1;b=2; { a=LEFT; echo $a >&2; } | { b=RIGHT; echo $b; tr 'A-Z' 'a-z'; }; echo "A=$a,B=$b"
LEFT
RIGHT
A=1,B=2
u-pl0:~/WORKDIR$ a=1;b=2; { a=LEFT; echo $a; } | { b=RIGHT; echo $b; tr 'A-Z' 'a-z'; }; echo "A=$a,B=$b"
RIGHT
left
A=1,B=2
```

7a. Will the following script count the number of lines in /etc/passwd?

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

0

7b. What about following?

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

nic

7c.

#(note: again, escape the variables (and the reverse apostrophes) in the HERE-document so they appear normally in the created file)

```
$ cat > count_lines.sh <<END
#!/bin/sh$

# One possible solution
count=0
while read x
do
   count=\`expr \$count + 1\`
done </etc/passwd
echo \$count
END
```

8. Write a script that changes the filename of the files in the current dir so that every filename is lowercased.

```
$ cat > lower.sh <<END
#!/bin/sh

# Make sure you are using POSIX locale
export LC_ALL=C

for f in *[A-Z]*
do
    if ls \$f 2>/dev/null; then
        mv "\$f" \`echo \$f | tr "[A-Z]" "[a-z]"\`
    fi
done
END
```

Poziční argumenty

Něco málo je zde: http://wiki.bash-hackers.org/scripting/posparams#mass_usage

1. **$0**
   The filename of the current script.
2. **$n**
   These variables correspond to the arguments with which a script was invoked. Here **n** is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
3. **$#**
   The number of arguments supplied to a script.
4. **$***
   All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
5. **$@**
   All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
6. **$?**
   The exit status of the last command executed.
7. **$$**
   The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
8. **$!**
   The process number of the last background command.

Výpis 2...n argumentů:

```
echo ${@:2}
```

Výpis posledního argumentu:

```
echo ${@: -1}
```

Výpis 1...n-1 argumentů:

```
length=$(($#-1))        # length je n-1
array=${@:1:$length}    # přiřadí od 1 do length
echo $array
```

Dosadit, pokud něco není inicializováno:

`${var:-5}` ... nastaví defaultní hodnotu na 5, pokud je prázdná

Test existence souboru:

```
#!/bin/sh$

if [ -e "a.txt" ]; then
    echo "Existuje."
else
    echo "Neexistuje."
fi
```

Zarovnání libovolného množství argumentů podle největšího:

```
#!/bin/sh$

params="$@"
length=0
for p in $params; do
      if [ ${#p} -gt $length ]; then
          length=${#p}
      fi
done
printf "%${length}d\n" "$@"
```

Počítání aritmetických výrazů:

```
#!/bin/sh$

a=`echo "$1" | grep -o '^[0-9]*'`
op=`echo "$1" | grep -o '[^0-9]'`
b=`echo "$1" | grep -o '[0-9]*$'`

vysl=`expr $a $op $b`
printf "%d\n" "$vysl
```

A shell script that takes n+1 parameters, first parameter is expr operator (escaped) the rest is sequence of numbers on which the operator is subsequentially applied (e.g. "./script.sh '+' 1 2 3" prints 6).

```
#!/bin/sh$

op=$1
vysl=$2
while [ "$3" ]
do
    vysl=`expr $vysl "$op" $3`
    shift
done
echo "$vysl"
```

Write a shell script that prints its parameters each on a separate line sorted alphabetically (ascending).

```
#!/bin/sh

text=$(echo $* | tr " " "\n" | sort)
echo "$text"
```

Test

```
[ "$a" = "$b" ]    ... true, když jsou řetězce stejné
[ "$a" != "$b" ]   ... true, když jsou řetězce různé
[ $a -eq $b ]      ... true, když je $a = $b (číselně)
[ $a -ne $b ]      ... true, když je $a ≠ rovno $b
[ $a -gt $b ]      ... true, když je $a > $b
[ $a -ge $b ]      ... true, když je $a ≥ $b
[ $a -lt $b ]      ... true, když je $a < $b
[ $a -le $b ]      ... true, když je $a ≤ $b

[ -e "$file" ]     ... soubor $file existuje
[ -d "$file" ]     ... soubor $file existuje a je adresář
[ -f "$file" ]     ... soubor $file existuje a je obyčejný soubor

[ -n "$str" ]      ... $str není prázdný řetězec
[ -z "$str" ]      ... $str je prázdný řetězec

[ ! expr ]         ... true, když je expr false
[ expr1 -a expr2 ] ... true, když jsou expr1 i expr2 true
[ expr1 -o expr2 ] ... true, když je expr1 nebo expr2 true
```

CP opačně: 1. argument je cíl:

```
#!/bin/sh
if [ "$#" -lt "2" ]; then  # málo argumentů, tzn. méně jak 3 (název, cíl, zdroj)
    echo "Missing some operands!!!"
elif [ "$#" -eq "2" ]; then  # možnost cíl, zdroj
    cp $2 $1
else  # možnost adresář, zdroje ...
    if [[ -d $1 ]]; then
        cp ${@:2} $1
    else
        echo "Target operand must be directory."
    fi
fi
```

Rotace podle delimiteru: (prý nebezpečně pomalé řešení)

Write a script which expects three parameters, a file name, a number 'n' and a character 'c'. The script then performs a circular shift of each input line (input is in the file specified in the first parameter). The lines are considered to be split into fields separated with character 'c' and the number of elements to shift is given in number 'n'. Output is written to STDOUT.

```
$ ./script.sh /etc/passwd 3 :

changes:
login:*:uid:gid:full name:home dir:shell

to the following form:
gid:full name:home dir:shell:login:*:uid
```

```
#!/bin/bash
# $1 - soubor; $2 - rotace o kolik; $3 - delimiter
while read line; do   #čtu z $1
    for i in `seq "1" "$2"`; do
        first=$(echo "$line" | cut -d"$3" -f1)
        rest=$(echo "$line" | cut -d"$3" -f2-)
        #line=\`paste -d"$3" "$rest" "$first"\`
        line="$rest$3$first"
    done
    echo "$line"
done <"$1"
```

3 Pokročilejší příklady:

Write a script that reads STDIN line by line, expecting two columns separated by " ". First column is in format 'N<op>M', where <op> is a single character binary operator and second column is a number.
The script then inserts column between the first and second column containing either "==" or "!=" depending on whether the equation holds, e.g:

```
(INPUT)
2+5 6
3*3 9
6-2 4

(OUTPUT)
2+5 != 6
3*3 == 9
6-2 == 4
```

```
$ cat > evaluator.sh <<END
#!/bin/sh

# TODO: Each line should be checked for correct input format
# We consider following operators: +, -, *, /
while read line; do
    eq=\`echo \$line | cut -d" " -f1\`
    result=\`echo \$line | cut -d" " -f2\`

    n=\`echo \$eq | sed 's/[^0-9].*\$//'\`
    m=\`echo \$eq | sed 's/^.*[^0-9]//'\`
    op=\`echo \$eq | sed 's/^.*\([^0-9]\).*\$/\1/'\`

    # We have to handle asterisk separately
    echo "\$op" | grep -q '^*$' > && res=\`expr \$n \* \$m\` || res=\`expr \$n \$op \$m\`

    [ \$result -eq \$res ] && echo "\$eq == \$result" || echo "\$eq != \$result"
# File descriptor 0 refers to STDIN
done <&0

END
```

Write a shell script that takes all the files from the current directory with the .jpg suffix and changes their name to a number (e.g. 001.jpg, 002.jpg...). The numbers are assigned by the alphabetical order of the files.
(Hint: generate the new filenames in a similar fashion as the numbers in 4)
(Hint2: take care of the files that are already in the output format - for example change the files like 001.jpg to x001.jpg first before renaming the files)

```
$ cat > rename_jpgs.sh <<END
#!/bin/sh
x=1
count=\`ls ./*.jpg | wc -l\`

# Tohle je pro pĹ™Ăpad, Ĺľe by tam byl nÄ›jakÄ› soubor s pĹ™Äponou *.jpg a
# Ă˘ĂšelnÄ›m nÄ›zvem, mohlo by to selhat.
for file in *.jpg
do
   mv "\$file" "x\$file"
done
for file in *.jpg
do
   mv "\$file" \`printf "%0\${#count}d\n" \$x\`.jpg
   x=\`expr \$x + 1\`
done
```

Write a script that behaves similarly as 'ls -l' but instead of writing a name of a file/directory in the last column, it prints the full path to the file/directory.

```
$ cat > my_ls.sh <<END
#!/bin/sh

# Get the base pathname
dir=\$1
if [ ! -d \$1 ]; then
    dir=\`dirname \$1\`
fi
```

```
cd \$dir
fulldir="\`pwd\`"
cd -

# Set newline as IFS
IFS="
"
for x in \`ls -l \$1\`; do
    # Print the first line normally
    echo \$x | grep ^total && continue

    # Prepare the full pathna (escape the special characters for sed)
    fullname="\$fulldir/\`echo \$x | tr -s " " | cut -d" " -f9\`"
    escaped=\`echo "\$fullname" | sed 's#/#--ESC_SLASH--#g'\`
    echo \$x | sed "s/\([0-9][0-9]:[0-9][0-9]\) [^ ]\+/\1 \$escaped/" | sed 's#--ESC_SLASH--#/#g'
done
```

Write a script that accepts lines from STDIN in the following format:
target : z1 z2 z3 .. zN
where target, z1, z2, z3 .. zN are filenames. The script then prints all "target" files which are older than any of the z1 .. zN files on the same line (z1 .. zN are dependencies of the target file). Do not use parameters '-nt', '-ot' of the test command since they are not in the standard).

```
#!/bin/sh
while read line; do
    target=$(echo "$line" | cut -d":" -f1 | tr -d " ")
    res=$(echo "$line" | cut -d":" -f2 | tr " " "\n" | xargs -I{} find {} -newer "$target" | wc -l)
    [ "$res" -gt 0 ] && echo "$target"
done
```

Kalendář - zjišťování měsíců, jenž mají pátek 13.:

```
#!/bin/bash
# $1 ... den v týdnu; $2 ... den v měsíci; $3 ... rok

let i=1
out=""
for i in 1 2 3 4 5 6 7 8 9 10 11 12; do
    cal -m $i $year | tail -n+3 | sed 's/   / ~ /g' | sed 's/  / /g' | cut d" " -f$1
    echo "-------------------------------------------"
done
```

Write a script that takes two number parameters: month and year, passes them to the "cal" command and prints the output according to the following example:

$ ./my_cal2.sh 2 2017

```
      1  2  3  4
11 10  9  8  7  6  5
12 13 14 15 16 17 18
25 24 23 22 21 20 19
26 27 28
```

For comparison, original cal:
$ cal 4 2017

```
   February 2017
Su Mo Tu We Th Fr Sa
      1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28
```

```
#!/bin/sh
# $1 ... měsíc; $2 ... rok

cal "$1" "$2" | tail -n+3 |
sed -e 's/   /~~ /g' -e 's/  / ~/g' -e 's/^ /~/gm' -e
's/30~/30 ~/' -e 's/31~/3$
awk '
{
    for (j=1; j<=NR; j++){
        if(NR%2 == 1)
            for (i=1; i <= NF; i++)
                printf("%s ", $i);
        else
            for (i=NF; i > 0; i--)
                printf("%s ", $i);
        printf("\n");
        next;
    }
}' |
tr '~' ' '
```

Eval

Spustí příkaz po expanzi.
Využití: Indexování pole

```
arr1
arr2     a[i]            eval "\$arr$i"
arr3
```

Načítání do pole: (skript 1002.sh)

```
#!/bin/sh

for i in `seq "1" "$#"`; do
    #echo "\$arr$i=\$$i"
    eval "arr$i=\$$i"
done

for i in `seq "1" "$#"`; do
    eval "echo "arg$i = \$arr$i""
done
```

```
u2-12:~/WORKDIR$ sh 1002.sh aaa bb 2 33
arg1 = aaa
arg2 = bb
arg3 = 2
arg4 = 33
```

Funkce

Deklarace:

```
function () {
    cmd1
    ...
    [return]
}
```

Volání:

```
function $var1 $var2
```

Trap

Při zabíjení procesů, ...
Příkazy `kill`

- `-N <pid>`
- `-t`
- `-9 <pid>` ... tvrdý kill (SIGKILL)

Signály:

- SIGINT — prostě kill (ctrl + c)
- SIGKILL — tvrdý kill (jenom UNIXový tvrďáci ho stihnou odchytit)
- SIGHUP — používaný při SSH, zadrží
- SIGSEGV — chyba při nedostatku paměti (vzniká např. při dereferenci mimo vyhrazenou paměť)

Vypsání typů killů: `kill -l`

Odhalení keyboard-interrupt při čtení standardního vstupu

```
#!/bin/bash
trap "echo Interrupted; sleep 5" SIGINT

while read line; do
    echo "$line"
    sleep 0.5
done
```

…

Awk

Formát:
/ ...hledaný pattern... / { ...co se má provést... }

- **Všechno musí být v bloku! Viz. vzory.**

Basic info:

- Vzory:
  - {...} vykoná blok pro všechny řádky (recordy)
  - BEGIN{...} provede se na začátku
  - END{...} provede se na konci
  - /reg vyraz/ { …}
  - (logicky vyraz) { … }
  - vzor, vzor { … }
    - Carka je operator rozsahu. awk zpracovava interaval radek, ktery zacina radkem vyhovujici prvnimu vzoru a konci radkem, ktery vyhovuji druhemu vzoru. Pote, co awk nalezne druhy vzor (dokonci zpracovavani rozsahu), opet zacina hledat prvni vzor.
- Myšlenkové formátování:
  - Řádky = **Records** (záznamy)
  - Objekty na řádkách = **Fieldy** (pole)
    - Defaultně se oddělují mezerou

- Odkazy na pole & proměnné:
  - **$n** n-té pole na aktuálním řádku (záznamu)
  - **$0** cely zaznam (cela radka) nebo aktualni zaznam (jako jedna promenna)
  - **NF** pocet poli v aktualnim zaznamu (`{printf $1, $NF}` vytiskne prvni a posledni pole oddelene mezerou)
  - **NR** cislo aktualniho zazamu
  - **FS** oddelovac vstupnich poli (implicitne mezera nebo tabulator)
  - **RS** oddelovac vstupnich zazamu (implicitne "novy_radek")
  - **OFS** oddelovac vystupnich poli (implicitne mezera)
  - **ORS** oddelovac vystupnich zazanamu (implicitne "novy radek")
  - **FILENAME** jmeno aktualniho vstupniho souboru
- Řídící struktury:
  - **break** ukoncuje cykl
  - **continue** zacina dalsi iteraci cyklu
  - **next** zacne se zpracovavat dalsi vstupni zaznam
    - Nejde volat v BEGIN a END
  - **exit** prejde se na vzor END

Vybira vsechny radky, ve kterych prvni pole odpovida regularnimu vyrazu ^[Aa]

```
awk '$1~/^[Aa]/ {print}'
```

Vybira radky, ve kterych, pokud je prvni sloupec mensi nez 5 a vetsi nez 1 nebo je vetsi nez 5 a vetsi nez 8. (Tj. napr. cisla 2,3,4,9,10)

```
awk '$1 < 5 ? $1 >1 : $1 > 8 {print}' cisla.txt
```

Výpis řádky v opačném pořadí:

```
for (i=NF; i > 0; i--)
    printf("%s ", $i);
```

Výpis každé 10. řádky:

```
#!/bin/sh
cat $1 |
awk '{
    if(NR%10 == 0){
        printf("%s\n", $0);
    }
    next;
}'
```

Výpis linků v html souboru:

```
#!/bin/sh
cat $1 |
awk '
BEGIN{
    tags=0;
    RS="<";
    #FS="";    # Konflikty mezer nenastávají, $0 se
vypíše včetně mezer
}
{
    for(i=0; i < NF; i++){
        if($0 ~ /^ *a /) {tags++};
        next;
        #print($0);
    }
    #printf("%s\n", tags);   # Tento vrátí 0
}
END{
    printf("%s\n", tags);   # Tento vrátí počet linků
}
'
```

Přidání sloupce se sumou předchozích:

Write an awk script which expects a file in the csv format at the input. This means that each line of the input line consists of several fields separated with a comma ",". The first line is the header line, it contains the column names. The first column of the subsequent lines contains a name, the rest of columns contain numbers. The script adds a new column called “Sum” (in the header line), which on each line contains the sum of the number fields. Exaple:

```
Input:

Name,pts1,pts2,pts3
Hynek,3,12,9
Jarmila,7,34,1
Vilém,8,27,0

The following output is produced by the awk script:

Name,pts1,pts2,pts3,Sum
Hynek,3,12,9,24
Jarmila,7,34,1,42
Vilém,8,27,0,35
```

```
#!/bin/sh
awk '
BEGIN{
    FS=",";
    OFS=","
}
NR==1{
    printf("%s,Sum\n", $0)
}
NR>1{
    Sum=0;
    for(i=1; i <= NF; i++){
        Sum+=int($i);
    }
    $(NF+1)=Sum;
    print($0);
}
'
```

Task for the last lab:

Write a program that is executed every midnight, check for the possible "vulnerabilities" in the system and sends mail to the admin.

These 4 "vulnerabilities" are checked:

1. Passwords
- check for empty user passwords (stored in /etc/shadow) with uid < 100 which are not listed in /sec/passwd
(choose a format you like, use /tmp/sec/passwd during practicals)
(for testing create a fake /etc/shadow file locally from /etc/passwd by substituting the second column by random strings)
- for such users set password to "disable" (in /etc/shadow)
- send message to the user (non-empty) if logged-in

2. Directory permissions
- in every home directory (except root's), find files starting with '.' (non-recursively) that:
-- have the same owner as the owner of the home directory and have a write permission for more than the owner
-- for such files, remove the write permission for everyone except the owner and send a report
- if the file has a different owner than the home dir, change the owner to the home dir owner and log a report

3. Passwords in ~/.netrc
- in every home directory, there is a ~/.netrc which contains information about the ftp servers in the format:

machine <machine_name>
login <login>
passwd <passwd>

machine
login
passwd

- There can be other non empty lines in between, however, those lines must be commented out (start with #). If you find invalid lines, report that the file has invalid format.
- login and password field can be missing
- The fields are separated by newline
- The entry blocks are separated by an empty line
- If the file contains server, which has a login different from "anonymous" and has a password (passwd category-value pair is present in the block), send a report and show all permissions of the file

4. New set[ug]id files
- Check all files in the system except directories and symlinks which have UID < 100 and have SetUID set and compare them with file /sec/setuid
-- The files that are not listed in the /sec/setuid are considered "new" and you should log a report about these files
-- Also, remove the SetUID for these "new" filesT

```
#!/bin/bash

#Bash comment block:
: <<'label'
blablabla
label

touch log.txt
today=$(date)
printf "Test run in %s\n" "$today" >> log.txt

#1. Passwords
# Predpokladam, ze /etc/shadow je soubor passwd (fake /etc/passwd).
# Uzivateli prijde zprava, pouze kdyz je prave zalogovany.
: <<'END'
rm /tmp/passwd
cat passwd | while read line; do
    expr1=$(echo $line | awk 'BEGIN{FS = ":";}{print $2;}')
    expr2=$(echo $line | awk 'BEGIN{FS = ":";}{print $3;}')
    if [ \( "$expr1" = "" \) -a \( "$expr2" -le "100" \) ]; then
        user=$(echo $line | grep -o '^[^:]*')
        preline=$(echo $line | grep -o '^[^:]*:')
        postline=$(echo $line | grep -o ':[^:]*:[^:]*:[^:]*:[^:]*:[^:]*$')
        printf "%sdisabled%s\n" "$preline" "$postline" >> /tmp/passwd
        printf "1: User %s dont have his password set.\n" "$user" >> log.txt
        echo "Set your password!!!" | write $user
    else
        echo "$line" >> /tmp/passwd
   fi
done
mv /tmp/passwd passwd
END
```

```
#2. Directory permissions
# Z neznameho duvodu to nic nedela, i kdyz casti kodu normalne funguji.
: <<'END'
ls -1 /home | while read user; do
    owner=$(ls -ld "/home/$line" | awk '{print $3}')    # vetsinou je to nazev slozky
    # Pres xargs je to asi rychlejsi, ale nemumim tam poustet vic prikazu.
    # find /home/$user -user "$owner" -perm /g+w,o+w | xargs -I{} chmod g-w o-w {}
    find /home/$user -user "$owner" -perm /g+w,o+w | while read line; do
        chmod g-w o-w $line
        printf "2: Disabling write perminitons to others than owner to %s\n" "$line" >> log.txt
    done
done
END

#3. Passwords in ~/.netrc
# U me tento soubor realne neexistuje, opet vytvorim fake: .netrc
: <<'END'
ls -1 /home | while read user; do
    fcontent=$(cat /home/$user/.netrc | awk '
        BEGIN{
            RS = "";    #// "" je specialni oznaceni prazdneho radku prave u RS.
            FS = "\n";
            invalidByFormat = 0;
            invalidByAnonymous = 0;
        }
        {
            if((NF == 1) && $1 ~ /^\s*machine .*/){
                printf("%s\n\n", $0);
            }
            else if((NF == 3) && ($1 ~ /^[ \t]*machine .*/) && ($2 ~ /^[ \t]*login .*/) && ($3 ~ /^[ \t]*password/)){
                if(($2 ~ /^[ \t]*login anonymous[ \t]*/) && ($3 ~ /^[ \t]*password ..*/))
                    invalidByAnonymous = 1;
                printf("%s\n\n", $0);
```

```
            }
            else{
                for(i=1; i <= NF; i++){
                    printf("#%s\n", $i);
                    invalidByFormat = 1;    #// Na konci cyklu se kdyztak zapise hlaska.
                }
                printf("\n");
            }
        }
        END{
             print invalidByFormat;
             print invalidByAnonymous;
        }
    '
    )
    isInvalidByFormat=$(echo "$fcontent" | tail -2 | head -1)    # Nejak mi nejde strcit ten obsah rovnou do testu :-(
    if [ $isInvalidByFormat -eq 1 ]; then
        echo "3: Invalid format of .netrc at /home/$user" >> log.txt
    fi
    isInvalidByAnonymous=$(echo "$fcontent" | tail -1)
    if [ $isInvalidByAnonymous -eq 1 ]; then
        echo "3: There is an password set for anonymous in .netrc at /home/$user" >> log.txt
    fi
    echo "$fcontent" | head -n -2 > /home/$user/.netrc
done
END
```