



Přednáška 5

Regulární výrazy. Filtry grep, sed a awk.

Katedra počítačových systémů FIT, České vysoké učení technické v Praze

©Jan Trdlička, 2011

*Příprava studijního programu Informatika je podporována projektem financovaným z
Evropského sociálního fondu a rozpočtu hlavního města Prahy.*

Praha & EU: Investujeme do vaší budoucnosti



- Matematický formalismus pro popis slov/vět jazyků.
- Množina regulárních výrazů $RV(\Sigma)$ nad konečnou neprázdnou abecedou Σ je nejmenší množina slov nad abecedou $\{a_1, a_2, \dots, a_n, \emptyset, \varepsilon, :, +, *, (,)\}$

která:

- obsahuje $\{\emptyset, \varepsilon\}$
- pro každý symbol a ze Σ platí, že $RV(\Sigma)$ obsahuje výraz a
- $\alpha, \beta \in RV(\Sigma) \Rightarrow (\alpha + \beta) \in RV(\Sigma)$
- $\alpha, \beta \in RV(\Sigma) \Rightarrow (\alpha \cdot \beta) \in RV(\Sigma)$
- $\alpha \in RV(\Sigma) \Rightarrow \alpha^* \in RV(\Sigma)$

• *Příklad:* $((a \cdot b) + ((c + d)^* \cdot e))$

- Teorie jazyků se uplatňuje nejen v lingvistice, ale i v teorii programovacích jazyků, počítačové grafice, pro popis a konstrukci HW, v biologii...



grep [přepínače] vzor [soubory]

- Implicitně vypíše na standardní výstup řádky, které obsahují zadaný vzor.
- **Vzor** může být definován **základním regulárním výrazem** (`man -s 5 regex`).
- Podporuje znaky: `., *, ^, $, \<, \>, \[, \], \{, \}`,
- Název **grep** je zkratka pro skupinu příkazů editoru **ex** (**g**lobally search for **r**egular expression and **p**rint result).
 - i nerozlišuje malá a velká písmena
 - v vypíše na standardní výstup řádky, které neobsahují zadaný vzor
 - c vypíše pouze počet odpovídajících řádek
 - l vypíše pouze jména souborů, které odpovídající řádky obsahují
 - n vypíše odpovídající řádky a jejich pořadové číslo v souboru



```
grep 'The' /etc/ssh/ssh_config
```

```
grep 'the' /etc/ssh/ssh_config
```

```
grep -i 'The' /etc/ssh/ssh_config
```

```
grep -ci 'the' /etc/ssh/ssh_config
```

```
grep -ni 'the' /etc/ssh/ssh_config
```

```
grep -l 'kill' /etc/init.d/*
```

```
grep root /etc/group
```

```
grep -v root /etc/group
```



Regulární výrazy – grep

- Pro definování vzoru se používají **speciální znaky**, jejichž význam je jiný než v shellu, musíme je před ním proto chránit (apostrofy, \).

Symbol	Význam
.	jeden jakýkoliv znak
znak*	žádný nebo libovolných počet výskytů předchozího znaku
[]	jeden znak z množiny/intervalu (např. [adf], [a-h])
[^]	jeden libovolný znak mimo znaků z množiny/intervalu
^	začátek řádky
\$	konec řádky
\<	začátek slova
\>	konec slova
\znak	ruší speciální význam následujícího znaku



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

```
ypcat passwd | grep '/bin/ksh$'
```

```
grep 'the' /etc/ssh/ssh_config
```

```
grep '\<the\>' /etc/ssh/ssh_config
```

```
grep 'bag' /usr/dict/words
```

```
grep '^bag' /usr/dict/words
```

```
grep 'bag$' /usr/dict/words
```

```
grep '^bag$' /usr/dict/words
```



```
grep '^b[aeiou]g' /usr/dict/words
```

```
grep '^b[^aeiou]g' /usr/dict/words
```

```
grep '^b.g$' /usr/dict/words
```

```
grep '^woo*' /usr/dict/words
```

```
grep '^wood' /usr/dict/words
```

```
grep '^wood.*d' /usr/dict/words
```

```
grep '^wood.*d$' /usr/dict/words
```



Regulární výrazy – grep

Symbol	Význam
<code>znak\{m\}</code>	právě m výskytů předchozího znaku
<code>znak\{m, \}</code>	nejméně m výskytů předchozího znaku
<code>znak\{m, n\}</code>	m až n výskytů předchozího znaku

Příklady:

```
grep '^[A-Z]' /usr/dict/words
```

```
grep '^[A-Z][A-Z]' /usr/dict/words
```

```
grep '^[A-Z]\{2\}' /usr/dict/words
```

```
grep '^[A-Z]\{2,3\}' /usr/dict/words
```




fgrep [přepínače] vzor [soubory]

- Implicitně vypíše na standardní výstup řádky, které obsahují zadaný vzor.
- **Vzor** může být definován pouze jako **obyčejný řetězec**.
- Příkaz je rychlejší než **grep** a **egrep**.
- Přepínače jsou podobné jak u příkazu **grep**.

Příklady:

```
fgrep 'root' /etc/group
```

```
fgrep '^root' /etc/group
```



egrep [přepínače] vzor [soubory]

- Implicitně vypíše na standardní výstup řádky, které obsahují zadaný vzor.
- **Vzor** může být definován **rozšířeným regulárním výrazem** (**man -s 5 regex**).
- Nepodporuje znaky: `\(, \), \n, \<, \>, \{, \}`
- Navíc podporuje znaky: `+, ?, |, (,)`
- Přepínače jsou podobné jako u příkazu **grep**.



Regulární výrazy – egrep

Symbol	Význam
znak+	jeden nebo libovolných počet výskytů předchozího znaku
znak?	žádný nebo jeden výskyt předchozího znaku
RE1 RE2	RE1 nebo RE2
(RE)	označení reg. podvýrazu RE

Příklady:

```
egrep '^wo+' /usr/dict/words
```

```
egrep '^wo?' /usr/dict/words
```

```
egrep 'work(out|man|shop)\' /usr/dict/words
```



`sed [přepínače] ' příkaz ' [soubory]`

`sed [přepínače] -f skript [soubory]`

- **S**tream **e**ditor - edituje neinteraktivně jeden nebo více souborů.
- Jednotlivé řádky jsou čteny ze standardního vstupu nebo ze souboru, provádí nad nimi jednotlivé příkazy a vypíší se na standardní výstup.

-n potlačí implicitní kopírování vstupu na výstup
(to co se má vytisknout musí být explicitně určeno příkazem **p** [print])

-f skript soubor skript musí obsahovat seznam příkazů:

`[adresa1 [,adresa2]] příkaz [parametry]`



```
$ cat data.txt
```

Jan	Novak	M	Praha	15000	26
Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32
Petra	Farska	Z	Praha	27000	27
Pavel	Kulik	M	Brno	24000	31

```
$ sed '' data.txt
```

Jan	Novak	M	Praha	15000	26
Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32
Petra	Farska	Z	Praha	27000	27
Pavel	Kulik	M	Brno	24000	31

```
$ sed -n '' data.txt
```



- **Příkazy**

d(delete)

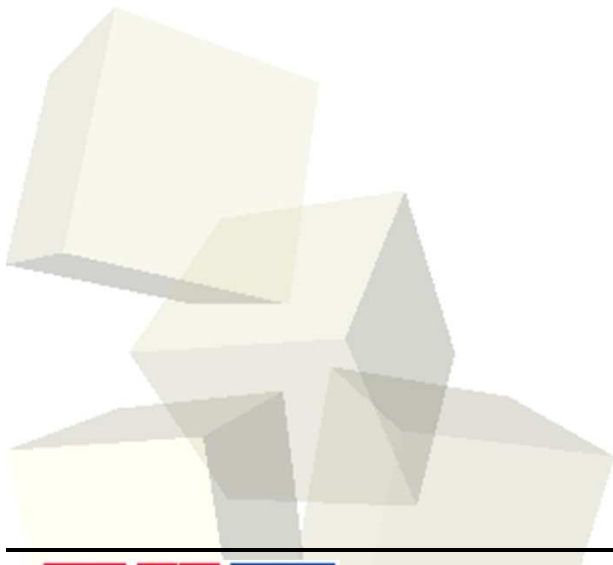
zruší řádku

p(print)

vypíše řádku na výstup

s/RE1/RE2/volby

nahradí text, který odpovídá vzoru RE1,
řetězcem RE2





```
$ sed -n '2,4p' data.txt
```

Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32
Petra	Farska	Z	Praha	27000	27

```
$ sed -n '4,$p' data.txt
```

Petra	Farska	Z	Praha	27000	27
Pavel	Kulik	M	Brno	24000	31



```
$ sed -n '/^J/p' data.txt
```

Jan	Novak	M	Praha	15000	26
Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32

```
$ sed '/^J/d' data.txt
```

Petra	Farska	Z	Praha	27000	27
Pavel	Kulik	M	Brno	24000	31

```
$ sed -n '/38$/ , /27$/p' data.txt
```

Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32
Petra	Farska	Z	Praha	27000	27



```
$ sed 's/Praha/Louny/' data.txt
```

Jan	Novak	M	Louny	15000	26
Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32
Petra	Farska	Z	Louny	27000	27
Pavel	Kulik	M	Brno	24000	31

```
$ sed 's/[0-9][0-9]$/& let/' data.txt
```

Jan	Novak	M	Praha	15000	26 let
Jiri	Prasek	M	Brno	22000	38 let
Jitka	Mala	Z	Plzen	23000	32 let
Petra	Farska	Z	Praha	27000	27 let
Pavel	Kulik	M	Brno	24000	31 let



`awk [přepínače] [program] [prom=hod...] [soubory]`

- Programovatelný filtr vytvořený autory: **A**ho, **W**einberger, **K**ernighan.
- Původní verze se jmenovala awk, rozšířená verze pak nawk (na některých systémech lze rozšířenou verzi najít pod původním názvem awk) .
- Jednotlivé řádky jsou čteny ze standardního vstupu nebo ze souboru, pokud odpovídají zadanému vzoru, provede se nad nimi příslušný program.
- Pokud není vzor specifikován, provede se program nad každou řádkou.
- Na řádku se pohlíží jako na posloupnost položek \$1, \$2,...,\$NF (\$0 = celá řádka).
- Implicitním oddělovačem položek je mezera/tabelátor (lze změnit přepínačem -F nebo proměnnou FS).
- Struktura příkazu programu:

`[vzor] [{ akce }]`



- **Typy vzorů:**

Vzor	Kdy se provede akce
BEGIN	před zpracováním první řádky ze vstupu
END	po zpracováním poslední řádky ze vstupu
výraz	pro řádky vyhovující danému výrazu
začátek,konec	od první řádky splňující výraz začátek až do první řádky splňující výraz konec

- **Typy výrazů:**

- regulární výraz (ve formátu pro egrep)
- logický výraz (0 nebo prázdný řetězec = false, jinak true)



- **Logické výrazy**

- tvořeny pomocí operátorů jako v jazyce C
- relační operátory: `>`, `>=`, `<`, `<=`, `==`, `!=`, `~`, `!~` (řetězec odpovídá/neodpovídá danému vzoru)
- matematické operátory: `+`, `-`, `*`, `/`, `%`, `^`, `++`, `--`
- Logické operátory: `&&`, `||`, `!`

- **Proměnné**

- deklarují se použitím
- použití je podobné jako v jazyce C

- **Některé předefinované proměnné**

- `$n` hodnota n-té položka z aktuálního řádku (`$0` = celá řádka)
- `NF` počet položek v aktuálním řádku
- `NR` pořadová číslo aktuální řádky
- `FS` vstupní oddělovač položek na řádce
- `OFS` výstupní oddělovač položek na řádce



```
$ awk '{print $2, $1}' data.txt
```

```
$ awk '{print $2 "\t" $1}' data.txt
```

```
$ ypcat passwd | awk -F: '{print $3 , $1 , $5}'
```

```
$ awk '/^J/ { print $0 }' data.txt
```

```
$ awk '{ printf("%d:  %s\n", NR, $0) }' data.txt
```



p1.awk

```
{ c=c+$5;
  print $0
}
END {
  printf("-----\n");
  printf("Prumerny plat                %d\n", c/NR)
}
```

`$ nawk -f p1.awk data.txt`

Jan	Novak	M	Praha	15000	26
Jiri	Prasek	M	Brno	22000	38
Jitka	Mala	Z	Plzen	23000	32
Petra	Farska	Z	Praha	27000	27
Pavel	Kulik	M	Brno	24000	31

Prumerny plat 22200



- **Podmíněný příkaz**

```
if (výraz) { příkazy1} [ else { příkazy2} ]
```

- **Cykly**

```
for ( i=min; i<=max; i++ ) { příkazy }
```

```
for ( j in pole ) { příkazy }
```

```
while ( výraz ) { příkazy }
```

```
do { příkazy } while ( výraz )
```

```
break      # předčasné ukončení cyklu
```

```
continue   # předčasné ukončení aktuální iterace cyklu
```



p2.awk

```
{  
    for (i=NF; i>=1; i--) { printf("%s\t", $i) }  
    printf("\n")  
}
```

\$ **nawk -f p2.awk data.txt**

26	15000	Praha	M	Novak	Jan
38	22000	Brno	M	Prasek	Jiri
32	23000	Plzen	Z	Mala	Jitka
27	27000	Praha	Z	Farska	Petra
31	24000	Brno	M	Kulik	Pavel



- Předdefinované funkce**

```
printf("řetězec" [,hodnoty])
```

```
sin(), sqrt(), log(), exp(), ...
```

```
system()
```

```
length(), match(), split(), substr(), sub(), ...
```

```
tolower(), toupper(), ...
```