Regular expressions.

Filters grep, sed a awk.

Department of Computer Systems FIT, Czech Technical University in Prague

grep

`grep [options] pattern [files]`

- Grep searches text files for a pattern and prints all lines that contain that pattern.
- **Pattern** can be defined by **limited regular expressions**

  (`man -s 5 regex`).
- Supported symbols: `.`, `*`, `^`, `$`, `\<`, `\>`, `\`, `[`, `]`, `\{`, `\}`,
- `grep` =**g**lobally search for **re**gular expression and **p**rint result

-i	Ignores upper/lower case.
-v	Prints all lines except those that contain the pattern.
-c	Prints only a count of the lines that contain the pattern.
-l	Prints only the names of files with matching lines
-n	Precedes each line by its line number in the file (first line is 1).

Examples

```
$ 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
```

Regular expressions – grep

- Regular expressions uses special characters, but their meanings is different from shell special characters.

Symbol	**Action**
`.`	Match any character.
`char*`	Match zero or more preceding character.
`[  ]`	Match one from a set/interval (e.g. [adf], [a-h]).
`[^ ]`	Match any except one from a set/interval.
`^`	Match beginning of line.
`$`	Match end of line.
`\<`	Match word’s beginning.
`\>`	Match word’s end.
`\char`	Escape character following.

Examples

```
$ 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
```

Examples

```
$ 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
```

Regular expressions – grep

Symbol	Action
`char\{m\}`	Match exactly n occurrences.
`char\{m,\}`	Match at least n occurrences.
`char\{m, n\}`	Match any number of occurrences between m and n.

Examples:

```
$ 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

```
fgrep [options] patter [files]
```

- The fgrep (fast grep) utility searches files for a character string and prints all lines that contain that string.
- It searches for a **string**, instead of searching for a pattern that matches an expression.
- It is more faster then `grep` and `egrep`.
- Similar options like `grep`.

Examples:

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

egrep

```
egrep [options] pattern [files]
```

- The egrep (expression grep) utility searches files for a pattern of characters and prints all lines that contain that pattern.
- egrep uses **full regular expressions** (`man -s 5 regex`).

Not supported symbols: `\(`, `\)`, `\n`, `\<`, `\>`, `\{`, `\}`.

New supported symbols: `+`, `?`, `|`, `(`, `)`.

- Similar options like `grep`.

Regular expressions – egrep

Symbol	Action
`char+`	Match one or more preceding.
`char?`	Match zero or one preceding.
`RE1 | RE2`	Separate choices to match.
`(RE)`	Group expressions to match.

Examples:

```
$ egrep '^wo+' /usr/dict/words
$ egrep '^wo?' /usr/dict/words
$ egrep 'work(out|man|shop)‘ /usr/dict/words
```

sed

```
sed [options]  ' commands '  [files]
sed [options] -f script [files]
```

- **S**tream **ed**itor - reads one or more text files, makes editing changes according to a script of editing commands, and writes the results to standard output.

-n	Suppresses the default output.
-f script	script contains list of commands:

```
[address1 [,address2]] commands [parameters]
```

Examples

```
$ 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
```

sed

- **Commands**

`d(delete)`	delete line
`p(print)`	print line to output
`s/RE1/RE2/options`	substitute the replacement string for instances of the regular expression in the pattern space.

Examples

```
$ 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
```

Examples

```
$ 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
```

Examples

```
$ 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, nawk, and gawk

```
awk [options] [prog] [variable=velue...] [files]
```

- Awk=**A**ho, **W**einberger, **K**ernighan.
- The awk utility scans each input file for lines that match any of a set of patterns specified in prog.
- The prog string must be enclosed in single quotes (') to protect it from the shell.
- For each pattern in prog there can be an associated action performed when a line of a file matches the pattern.
- Input line consists of items $1, $2,...,$NF ($0 = whole line).
- DefaultIt field separator is space/TAB (can be change by option –F or by variable FS).
- Structure of awk script:

```
[ pattern ] [{ action }]
```

awk, nawk, and gawk

- **Types of pattern**:

Pattern	**When the action is executed**
`BEGIN`	Before the first line from input
`END`	After the last line from input
`expression`	For all lines which satisfies the expression
`begin,end`	From the first line satisfies the expression `begin` until the first line satisfies the expression `end`

- **Type of expression**:
  - Regular expression (the same like egrep)
  - Logical expression (0 or empty string = false, else true)

awk, nawk, and gawk

- **Logical expressions**
  - Usage like in C language
  - Relational operators: `>`, `>=`, `<`, `<=`, `==`, `!=`, `~`, `!~`
  - Mathematical operators: `+`, `-`, `*`, `/`, `%`, `^`, `++`, `--`
  - Logical operators: `&&`, `||`, `!`
- **Variables**
  - Usage like in C language
- **Built-in variables**

`$n`	n-th field in the current record (`$0` = the whole line)
`NF`	number of fields in current record
`NR`	number of the current record
`FS`	field separator (default is blank)
`OFS`	output field separator

Examples

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

$ nawk '{print $2 ”\t” $1}' data.txt

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

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

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

Examples

p1.awk

```
{ c=c+$5;
  print $0
}
END {
  printf("----------------------------------------\n");
  printf("Average salary                   %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
------------------------------------------
Average salary               22200
```

awk, nawk, and gawk

- **Conditional statement**

```
if (expression) { cmd1} [ else { cmd2} ]
```

- **Loops**

```
for ( i=min; i<=max; i++ ) { cmd1; cmd2; … }
for ( j in array ) { cmd1; cmd2; … }
while ( expression )  { cmd1; cmd2; … }
do { cmd1; cmd2; … } while ( expression )
break       # exit from a loop
continue    # begin next iteration of loop  without reaching
              the bottom.
```

Examples

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
```

awk, nawk, and gawk

- **Built-in functions**

```
printf("string" [,values])

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

system()

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