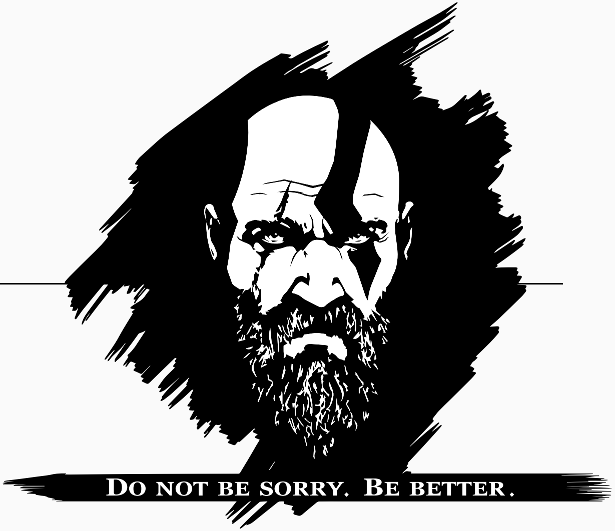
42sh - Presentation

ACU 2019 Team



Copyright

Introduction

Unix

- Learn about the internals of a shell
- Improve your skills in Unix systems
- Learn to write a manpage and proper documentation

Programming

- Use everything you learned in the C language
- Discover a high-level scripting language

Project and team management

- Learn to work as a team
- Learn to think and design your project
- Prove your worth or get back on tracks
- Work on a real project

42sh is a project that covers a lot of ground:

- Unix shell
- Unit testing
- Documentation
- Advanced build systems
- Design thinking
- Team and project management

On the technical side

- A clean and close to perfect binary
- Ability to conform to a reference documentation
- Clean C99 code and compliance to the coding style
- Correct use of git
- Real unit testing

On the management side

- Team management and handling of social problems
- Tasks assignment

SCL

- SCL is a SUS chapter
- This documentation defines the standard Unix shell behaviors
- It's your bedside book
- It will tell you how to handle expressions such as:

```
42sh$ e"$(echo ch)o" a
-> tokens: 'e"$(echo ch)o"' and 'a'

42sh$ e"$(echo ch)o "a
-> tokens: 'e"$(echo ch)o "a'

42sh$ e`echo "cho "`a
-> tokens: 'e`echo "cho "`a'
```

Project

Overall

A shell is composed of 3 main parts:

- Lexer
- Parser
- Execution

Lexer

- Check the lexical validity of the input
- Split raw input into token
- Recognize keywords and key symbols

Lexer example

```
if pwd && ls; then
    cd;
else
    alias;
fi
```

- Can for example be converted in:
  IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

- A set of rules
- Describe the form, not the meaning

Shell grammar

Grammar

```
input:       list EOF
        |    EOF

list:        and_or ((';'|'&') and_or)* [';'|'&']

and_or:      shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |    rule_if
        |    command

rule_if:     'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |    'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:       list EOF
        |    EOF

list:        and_or ((';'|'&') and_or)* [';'|'&']

and_or:      shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |    rule_if
        |    command

rule_if:     'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |    'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |     EOF

list:         and_or ((';'|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |     rule_if
        |     command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |     'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:       list EOF
          |  EOF

list:        and_or ((';'|'&') and_or)* [';'|'&']

and_or:      shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
          |  rule_if
          |  command

rule_if:     'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
          |  'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:          list EOF
        |       EOF

list:           and_or ((';'|'&') and_or)* [';'|'&']

and_or:         shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |       rule_if
        |       command

rule_if:        'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |       'elif' list 'then' list [else_clause]
```

Shell grammar

IF -> WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |     EOF

list:         and_or ((';'|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |     rule_if
        |     command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |     'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |     EOF

list:         and_or ((';'|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |     rule_if
        |     command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |     'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |     EOF

list:         and_or ((';'|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |     rule_if
        |     command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |     'elif' list 'then' list [else_clause]
```

Shell grammar

LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |   EOF

list:         and_or ((';'|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |     EOF

list:         and_or ((';'|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |     rule_if
        |     command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |     'elif' list 'then' list [else_clause]
```

Shell grammar

LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell\_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

LOGICAL_AND -> WORD -> SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:       list EOF
        |    EOF

list:        and_or ((';'|'&') and_or)* [';'|'&']

and_or:      shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |    rule_if
        |    command

rule_if:     'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |    'elif' list 'then' list [else_clause]
```

Shell grammar

SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:        list EOF
        |     EOF

list:         and_or (('; '|'&') and_or)* [';'|'&']

and_or:       shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |     rule_if
        |     command

rule_if:      'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |     'elif' list 'then' list [else_clause]
```

Shell grammar

SEMICOLON -> THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:       list EOF
        |    EOF

list:        and_or ((';'|'&') and_or)* [';'|'&']

and_or:      shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |    rule_if
        |    command

rule_if:     'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |    'elif' list 'then' list [else_clause]
```

Shell grammar

THEN -> WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:      list EOF
        |   EOF

list:       and_or ((';'|'&') and_or)* [';'|'&']

and_or:     shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |   rule_if
        |   command

rule_if:    'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |   'elif' list 'then' list [else_clause]
```

Shell grammar

WORD -> SEMICOLON -> ELSE -> WORD -> SEMICOLON -> FI

Grammar

```
input:       list EOF
        |    EOF

list:        and_or ((';'|'&') and_or)* [';'|'&']

and_or:      shell_command (('&&'|'||') ('\n')* shell_command)*

shell_command:  '{' list '}'
        |    rule_if
        |    command

rule_if:     'if' list 'then' list [else_clause] 'fi'

else_clause:    'else' list
        |    'elif' list 'then' list [else_clause]
```

Parser

- Check the syntax validity of the tokens list
- Construct an AST

Parser example

```
if CONDITION then
    BODY
else
    ELSE_BODY
fi
```

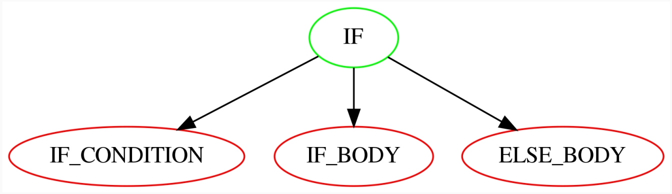


Figure 1: An AST example

Parser example

```
if CONDITION then
    cd
else
    alias
fi
```

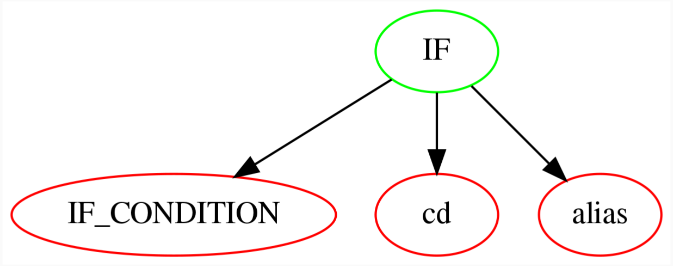


Figure 2: An AST example

Parser example

```
if pwd && ls then
    cd
else
    alias
fi
```

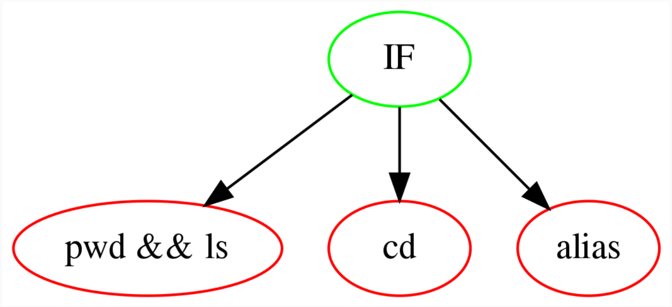


Figure 3: An AST example

Parser example

```
if BIN_OP then
    cd
else
    alias
fi
```

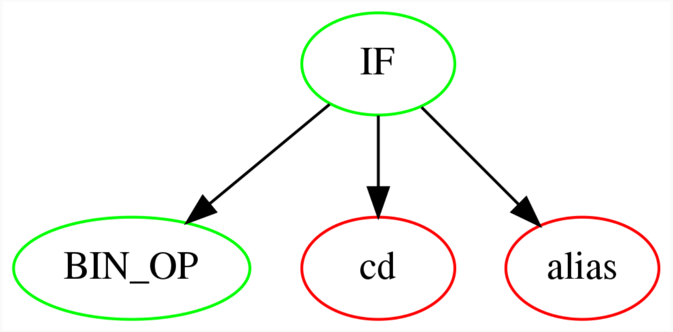


Figure 4: An AST example

Parser example

```
if BIN_OP(LEFT, OP, RIGHT) then
    cd
else
    alias
fi
```

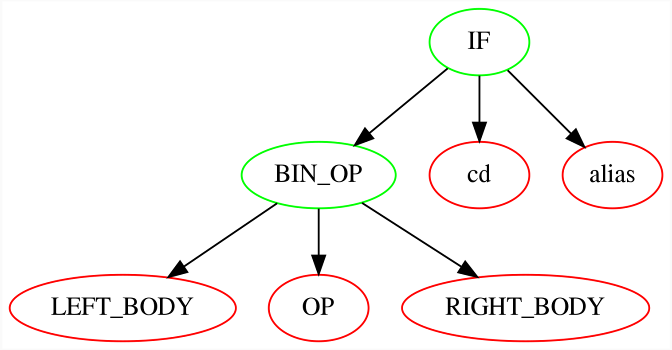


Figure 5: An AST example

Parser example

```
if pwd && ls then
    cd
else
    alias
fi
```

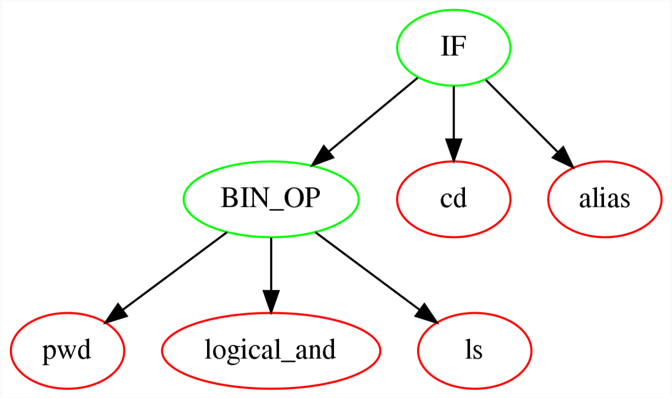


Figure 6: An AST example

Execution

- Interpret your AST
- Execute the command line
- Handle fork/exec and redirection

v0.3

- man page
- build system
- testsuite
- simple execution

v0.5

- Improve execution
- Your shell execution should be working at 100%

v0.8

- tilde expansion
- builtins
- variables
- quoting
- path expansion
- arithmetic expansion
- aliases

v0.9

- command substitution

v1.0

- git sanity
- zero leaks
- zero segfaults
- your 42sh is now perfect

Oral examinations

- You will have 4 defenses:
  - v0.3
  - v0.5
  - v0.8
  - v1.0

v0.3, v0.5 and v0.8

- Duration - 15 min
- Advice for the next version
- Malus tracking: memory leaks, segmentation faults, etc
- Feedback about your work

Last oral (v1.0)

- Duration - 30 min
- Presentation of the whole project
- Test modules that can be evaluated manually
- Test your global knowledge about the project
- Malus tracking: memory leaks, segmentation faults, etc
- Feedback about your work

- Do not mix up *definitly lost* and *still reachable* categories in valgrind
- Memory leak is about an amount of memory that grows up without control, slowly but/or surely, because objects (whose pointers have been lost) have not been freed
- *Still reachable* are the only authorized leaks
- To quote Stack-Overflow: it is like rearranging the deck chairs while the ship sinks around you

Timeline

Dementors and code review

- Only if you deserve it!
- If you do not, we will cancel them!

v0.3

- v0.3 - 4 days - November 12th
- **Dementor:** November 14th
- **Final:** November 16th

v0.5

- v0.5 - 13 days - November 16th
- **Dementor:** November 26th
- **Final:** November 29th

v0.8

- v0.8 - 8 days - November 29th
- **Dementor:** December 5th
- **Final:** December 7th

v0.9 and v1.0

v0.9 - 1 days - December 7th

- **Final:** December 8th

v1.0 - 2 days - December 8th

- **Final:** December 10th

- Shell Command Language exam
- Defenses
- Mandatory modules
- Advanced modules

Be efficient

- Start as soon as possible
- Ask for help... Do not stay stuck...
- Think and design, *then* start coding
- “Premature optimization is the root of all evil” – Donald Knuth
- *Use Git*
- Respect the coding style from the beginning
- Do not cheat

Tools

- Any language you wish for your testsuite
- If it’s not on the PIE, ask with a news first.
- Unit test will be highly considered
- You can use ceedling for your unit tests, or any tool on the PIE.

How

```
cat script.sh | ./42sh

./42sh < script.sh

./42sh -c 'command'

./42sh script.sh

./42sh
```

Conclusion

How to fail

You should *NOT* split work as follow:

- One member on the lexer
- One member on the parser
- One member on the execution
- One member on the testsuite

How to (maybe) succeed

- Create the simplest shell
- Split by features (if, pipe, etc)
- Each member can implement a feature, from lexer to execution, with the corresponding test

Newsgroup: `assistants.projets`, `[42SH]` tag.

Group of: 4

As usual:

- Your project must comply with the coding style.
- Cheating will be sanctioned.
- You will not be helped if you don't have a working build system or you didn't attempt debug.
- Use gdb, valgrind, libasan!
- Do not start to code now, think about your design!

Any questions?