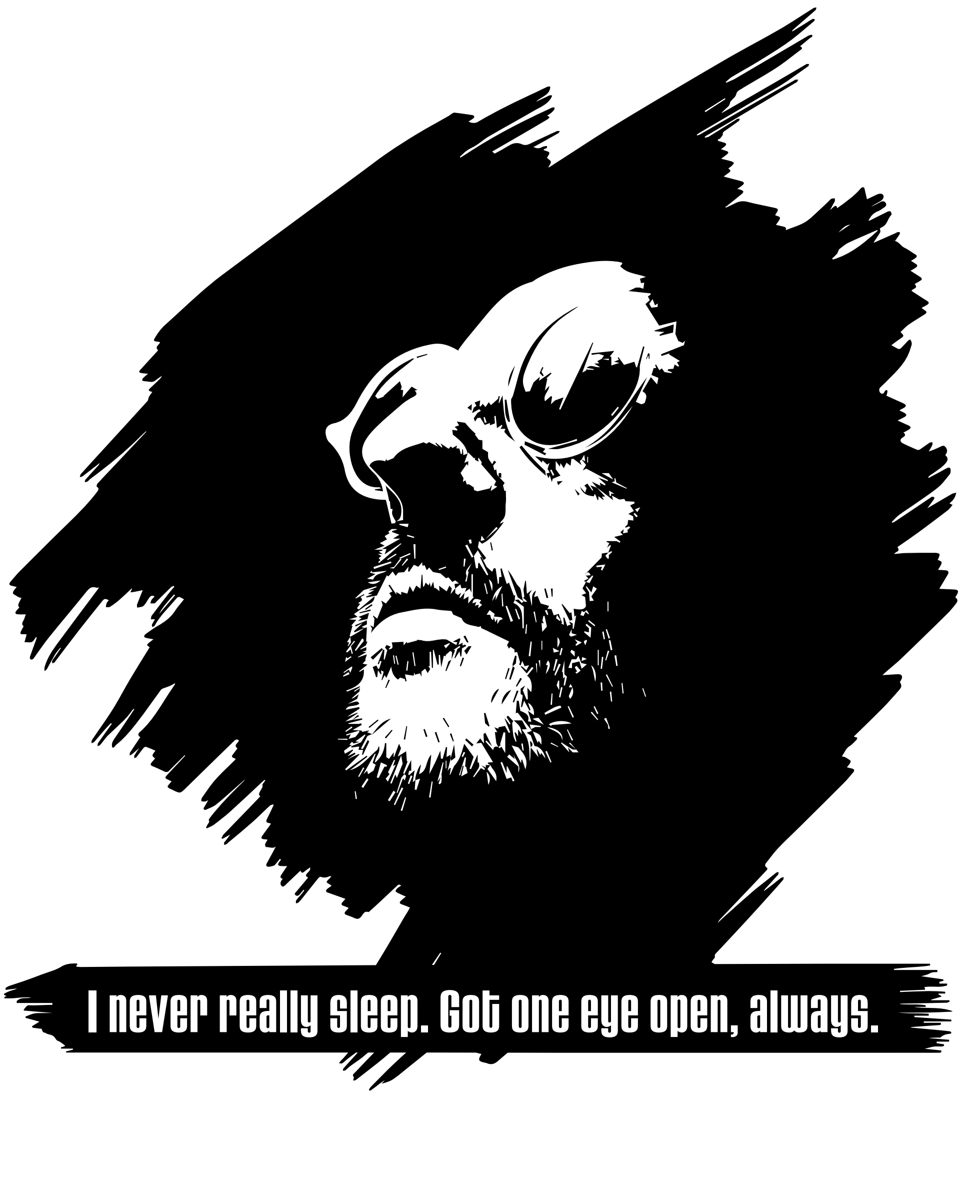
myFind - Apping — Subject

version **#1.0**



Assistants C/Unix 2020 `<assistants@tickets.assistants.epita.fr>`

Copyright

Contents

*https://intra.assistants.epita.fr

Obligations

*Obligations are **fundamental** rules shared by all subjects. They are non-negotiable and to not apply them means to face sanctions. Therefore, do not hesitate to ask for explanations if you do not understand one of these rules.*

Obligation #0: **Cheating**, as well as sharing source code, tests, test tools or coding-style correction tools is **strictly forbidden** and penalized by not being graded, being flagged as a cheater and reported to the academic staff.

Obligation #1: If you do not submit your work before the deadline, it will not be graded.

Obligation #2: Your submission repository must be **clean**. Except for special cases, which (if any) are **explicitely** mentioned in this document, an *unclean* repository may contain:

- binary files;[1]
- files with inappropriate privileges;
- forbidden files: `*~`, `*.swp`, `*.o`, `*.a`, `*.so`, `*.class`, `*.log`, `*.core`, etc.;
- a file tree that does not follow our specifications (see **??**).

Obligation #3: All your files must be encoded in ASCII or UTF-8 without BOM.

Obligation #4: When examples demonstrate the use of an output format, you must follow it scrupulously.

Obligation #5: Your code must compile with the flags:

```
-pedantic -Werror -Wall -Wextra -std=c99
```

Obligation #6: If a function, a command or a library is not *explicitly* authorized, it is **forbidden**. If you use a forbidden symbol, your work will not be graded.

Advice

- ▷ Read the *whole* subject.
- ▷ If the slightest project-related problem arise, you can get in touch with the assistants.

  Post to the dedicated **newsgroup** (with the appropriate **tag**) for questions about this document, or send a **ticket** to **<assistants@tickets.assistants.epita.fr>** otherwise.
- ▷ In examples, `42sh$` is our prompt: use it as a reference point.
- ▷ Do **not** wait for the last minute to start your project!

[1]If an executable file is required, please provide its sources **only**. We will compile it ourselves.

Project data

Instructors:

- LIONEL AUROUX `<lionel@assistants.epita.fr>`

Dedicated newsgroup: `assistants.projets` with `[FIND]`

Members per team: 1

1 Getting started

myfind is a simplified version of `find(1)`.

1.1 Instructions

Your goal is to write a program whose behavior follows `find(1)`.

The program's binary name must be `myfind` and must be generated at the root of your repository by your Makefile when the rule `all` is used.

In addition to your main assignment, you have to implement a test suite (in your `./tests/` directory).

Do not underestimate the test suite, it will allow you to ensure previously implemented features are still working (called regression testing[1]).

1.2 Goals

With myfind, you will learn how to read files metadata and recursively look through directories using the Unix API.

You will implement a simple command line parsing and an AST that will help you evaluate search expressions. Finally you will discover how to fork and execute some commands while your program is running.

[1] https://en.wikipedia.org/wiki/Regression_testing

2 Subject Rules

Files to submit:

- `./Makefile`
- `./src/*`
- `./tests/*`

Makefile: Your makefile should define at least the following targets:

- all: Produces the myfind binary
- check: Run your testsuite with myfind binary
- clean: Delete everything produced by make

Forbidden functions: You can use all the functions of the standard C library except:

- glob(3)
- regexec(3)
- wordexp(3)
- system(3)
- popen(3)
- syscall(2)
- ftw(3)
- nftw(3)
- fts_open(3)
- fts_read(3)
- fts_children(3)
- fts_set(3)
- fts_close(3)

3 Core features

This project will be evaluated so that implementing properly all the base features is barely sufficient to pass. You will need to implement some additional features to get a decent grade.

3.1 Error handling

For this project, if you encounter an error while parsing the command line, you must write an *explicit* message on `stderr` and return 1.

If you encounter an error while processing files and directories (with expression), you must write an *explicit* message on `stderr` and continue execution. When finished, your program must return 1.

Otherwise, your program must return 0.

Tips

You can use `err` and `warn` to properly print error and warning messages.

3.2 Basic Find

Tips

Look at `opendir(3)`, `readdir(3)` and `closedir(3)` functions.

You will first need a basic program, with the simplest usage of myfind: printing files. Myfind must be able to take zero or several arguments and print recursively the filename and the content of the directory given as argument. Arguments are processed from left to right. You must not follow symbolic links.

Examples:

```
42sh$ ls *
myfind

foo:
bar

qux:
baz
42sh$ ./myfind
.
./myfind
./foo
./foo/bar
./qux
./qux/baz
42sh$ cd foo; ../myfind ../
../
../myfind
../foo
../foo/bar
```

(continues on next page)

(continued from previous page)

```
../qux
../qux/baz
42sh$ cd -; ./myfind foo qux
foo
foo/bar
qux
qux/baz
42sh$ ln -s foo toto
42sh$ ./myfind toto
toto
```

3.3 Expressions

Your will now need to handle zero or more expressions. You must print the path when the expression returns *true*.

Your program must parse command lines of the following format:

```
42sh$ ./myfind [options] [files] [expressions]
```

Expressions must be applied to every file.

An expression can be one of the following types:

- test
- action
- operator

3.3.1 Name test

Tips

Look at `fnmatch(3)` function.

`-name` takes a parameter and returns true if the current filename matches the parameter. You must handle globbing.

Example:

```
42sh$ ./myfind foo
foo
foo/bar
foo/baz
42sh$ ./myfind foo -name bar
foo/bar
42sh$ ./myfind foo -name foo
foo
42sh$ ./myfind foo -name 'ba?'
foo/bar
foo/baz
```

3.3.2 Type test

Tips

Look at `readdir(3)` function and `stat(2)` syscall.

`-type` takes a parameter and returns true if the type of the current file matches the parameter. You must handle the following types:

- *b*: special files in block mode
- *c*: special files in character mode
- *d*: directories
- *f*: regular files
- *l*: symbolic links
- *p*: named pipes (FIFO)
- *s*: sockets

Example:

```
42sh$ ls *
myfind

foo:
bar

qux:
baz
42sh$ ./myfind -type d
.
./foo
./qux
```

Be careful!

Be careful if you decide to use the `d_type` field in `struct dirent`: you must properly handle `DT_UNKNOWN` case using `stat(2)`. For more information, please refer to the `readdir(3)` man page.

3.3.3 Or operator

`-o` is an operator placed between two expressions `expr1 -o expr2`. If expr1 is evaluated to true, expr2 will not be executed.

`expr1 -o expr2` is equivalent to one expression.

Example:

```
42sh$ ./myfind foo
foo
foo/baz
foo/bar
```

(continues on next page)

(continued from previous page)

```
42sh$ ./myfind foo -name bar
foo/bar
42sh$ ./myfind foo -name bar -o -name baz
foo/baz
foo/bar
```

3.3.4 And operator

`-a` is an operator placed between two expressions `expr1 -a expr2`. If expr1 is evaluated to false, expr2 will not be executed.

`expr1 -a expr2` is equivalent to one expression.

Example:

```
42sh$ ./myfind foo
foo
foo/baz
foo/bar
42sh$ ./myfind foo -name bar
foo/bar
42sh$ ./myfind foo -name bar -a -name baz
42sh$
```

The default operator between two expressions is `and`, which is why the next expression will not output anything.

```
42sh$ ./myfind foo -name bar -name baz
42sh$
```

3.3.5 Newer test

Tips

Look at `stat(2)` syscall, especially `st_mtime` field of `struct stat`.

`-newer` takes a file as parameter and returns true if the currently-examined file has a last modification date more recent than the file given as argument.

If the file is a symbolic link and the -H option or the -L option is in effect, the modification time of the file it points to is always used.

Example:

```
42sh$ mkdir foo
42sh$ touch foo/bar
42sh$ touch foo/baz
42sh$ find foo/* -newer foo/bar
foo/baz
42sh$ find foo/* -newer foo/baz
42sh$
```

Be careful!

You have to handle nanosecond-level differences: refer to `st_mtim.tv_nsec` field of `struct stat`.

4 Additional features

4.1 Myfind options

Your program must be able to parse options before files. It must parse command lines following this format:

```
42sh$ ./myfind [options] [files]
```

You have to implement the following options:

- `-d`: *myfind* should process each directory's content before the directory itself. This option follows the [BSD-family find](https://man.openbsd.org/find.1), and not GNU-find, where *-d* is not considered as an option, but as an expression always evaluating to true. By default, myfind visits directories in pre-order (before their contents).
- `-H`: *myfind* does not follow symbolic links, except while processing command line arguments.
- `-L`: *myfind* follows symbolic links.
- `-P`: *myfind* never follows symbolic links. This is the default behavior.

Examples:

```
42sh$ ls foo
bar baz
42sh$ ./myfind foo
foo
foo/bar
foo/baz
42sh$ ./myfind -d foo
foo/bar
foo/baz
foo
```

```
42sh$ ls
foo myfind
42sh$ ls foo
bar baz
42sh$ ln -s foo qux
42sh$ ./myfind qux
qux
42sh$ ./myfind -H qux
qux
qux/bar
qux/baz
```

```
42sh$ ls
qux foo myfind
42sh$ file qux
qux: symbolic link to foo
42sh$ ./myfind .
.
./myfind
```

(continues on next page)

(continued from previous page)

```
./foo
./foo/bar
./foo/baz
./qux
42sh$ ./myfind -L .
.
./myfind
./foo
./foo/bar
./foo/baz
./qux
./qux/bar
./qux/baz
```

4.2 Exec-family actions

myfind must be able to execute commands on matched files, using the following actions: `-exec`, `-execdir` and `-exec ... +`.

4.2.1 Exec

The `-exec` action executes the command passed by argument, delimited by `;`. Every `{}` string encountered in the command must be replaced with the current filename. It returns `true` if the command returned 0, `false` otherwise.

Examples:

```
42sh$ pwd
/tmp
42sh$ ./myfind foo
foo
foo/bar
foo/baz
42sh$ ./myfind foo -exec pwd \; -exec echo -- {} -- \;
/tmp
-- foo --
/tmp
-- foo/baz --
/tmp
-- foo/bar --
42sh$ ./myfind foo -exec md5sum {} \; -exec echo ok \;
md5sum: foo: Is a directory
d41d8cd98f00b204e9800998ecf8427e  foo/baz
ok
d41d8cd98f00b204e9800998ecf8427e  foo/bar
ok
```

4.2.2 Execdir

`-execdir` behavior is similar to `-exec`, except it executes the command in the current file's directory. `{}` placeholders will be replaced by the current file name, preceded by `./` (relative file path).

Examples

```
42sh$ ./myfind foo -execdir pwd \; -execdir echo -- {} -- \;
/tmp
-- ./foo --
/tmp/foo
-- ./baz --
/tmp/foo
-- ./bar --
```

4.2.3 Exec +

The `-exec command {} +` action is similar to the `-exec` action. The main difference is that the command line is built by appending the matching filenames at the end. Thus, instead of invoking the command once for each matched file, it is executed once for many of them at the same time.

The command line must end with `{}`. Any other instance of this placeholder in the command line will result in an error.

Examples

```
42sh$ pwd
/tmp
42sh$ ./myfind foo
foo
foo/baz
foo/bar
42sh$ ./myfind foo -exec echo {} \;
foo
foo/baz
foo/bar
42sh$ ./myfind foo -exec echo {} \+
foo foo/baz foo/bar
42sh$ ./myfind foo -exec echo {} a \+
myfind: missing argument to `-exec'
42sh$ echo $?
1
42sh$ ./myfind foo -exec echo {} {} \+
myfind: only one instance of {} is supported with -exec ... +
42sh$ echo $?
1
```

4.2.4 Resources Leaks

You shall close every file descriptor that you opened before executing the command (which means everything but `STDIN_FILENO`, `STDOUT_FILENO` and `STDERR_FILENO` from `unistd.h`).

You shall not leave zombie processes after *myfind* exits (c.f. `waitpid(2)`).

4.3 Expressions

4.3.1 AST

You will implement an AST.

In this additional part for expressions, you must implement the `!` and `()` operators. You must also implement operator precedence. The expression `-name toto -o -name tata -a -type f` is now equivalent to `-name toto -o ( -name tata -a -type f )`.

4.3.2 Not operator

`!` is an operator than can be placed before an expression `! expr`. It returns `true` if expr is false, and vice-versa.

`! expr` is equivalent to one expression.

Examples:

```
42sh$ ./myfind foo
./foo
./foo/baz
./foo/bar
42sh$ ./myfind -name bar
./foo/bar
42sh$ ./myfind '!' -name bar
.
./myfind
./foo
./foo/baz
```

4.3.3 Parentheses operator

You must handle parentheses. A pair of parentheses is an expression that returns true if the wrapped expressions returned true, false otherwise.

`( expr )` is equivalent to one expression.

Examples:

```
42sh$ ./myfind foo
./foo
./foo/baz
./foo/bar
42sh$ ./myfind \( -name bar -o -name baz \) -print
./foo/baz
./foo/bar
42sh$ ./myfind \! \( -name bar -o -name baz \) -print
./foo
```

4.3.4 Perm test

`-perm` takes a mode in octal as parameter and returns true if the current file's permission bits match exactly the mode.

`-perm -` takes a mode in octal as parameter and returns true if all the permission bits in mode are set for the file.

`-perm /` takes a mode in octal as parameter and return true if any of the current file's permission bits is set in mode.

Examples:

```
42sh$ ls -l foo
total 0
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 bar
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 baz
42sh$ ./myfind -perm 644
./foo/baz
./foo/bar
42sh$ chmod 123 foo/baz
42sh$ ./myfind -perm 644
./foo/bar
```

```
42sh$ ls -l foo
total 0
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 bar
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 baz
42sh$ ./myfind -perm -640
.
./myfind
./foo
./foo/baz
./foo/bar
42sh$ chmod 123 foo/baz
42sh$ ./myfind -perm -102
./foo/baz
```

```
42sh$ ls -l foo
total 0
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 bar
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 baz
42sh$ ./myfind -perm 644
./foo/baz
./foo/bar
42sh$ chmod o-r foo/baz
42sh$ ./myfind foo -perm /004
foo
foo/bar
```

4.3.5 User and Groups tests

`-user` takes a `username` and returns true if the file is owned by the user `username`.

`-group` takes a `groupname` and returns true if the file belongs to the group `groupname`.

`-user` and `-group` are tests.

Examples:

```
42sh$ ls -l foo
-rw-r--r-- 1 login_x login_x 0 oct.  3 20:42 bar
-rw-r--r-- 1 login_x test    0 oct.  3 20:42 baz
-rw-r--r-- 1 toto    test    0 oct.  3 20:42 qux
42sh$ ./myfind foo/* -user login_x
foo/baz
foo/bar
42sh$ ./myfind foo/* -group test
foo/qux
foo/baz
```

4.3.6 Print action

`-print` prints the path of the currently examined file. It always returns true.

Examples:

```
42sh$ ./myfind foo
foo
foo/baz
foo/bar
42sh$ ./myfind foo -print
foo
foo/baz
foo/bar
42sh$ ./myfind foo -print -a -print
foo
foo
foo/baz
foo/baz
foo/bar
foo/bar
42sh$ ./myfind foo -print -o -print
foo
foo/baz
foo/bar
```

4.3.7 Delete action

`-delete` deletes files and returns true if it succeeded. If the deletion fails, an error message is displayed.

Use of `-delete` implies use of `-d`.

Examples:

```
42sh$ ./myfind
.
./myfind
./foo
./foo/baz
./foo/bar
42sh$ ./myfind foo -name bar -delete -delete
myfind: cannot delete ‘foo/bar’: No such file or directory
42sh$ ./myfind
.
./myfind
./foo
./foo/baz
42sh$ ./myfind foo -delete
42sh$ ./myfind
.
./myfind
```

I never really sleep. Got one eye open, always.