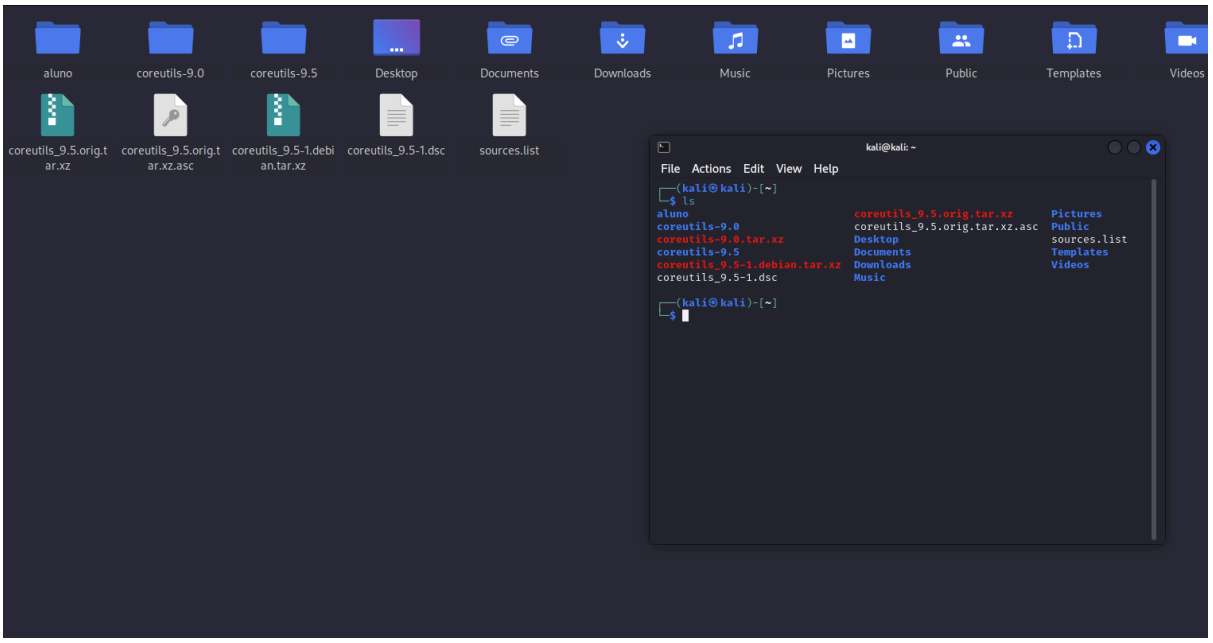
Trabalho 1 - Relatório sobre modificação do comando “ls” do Linux

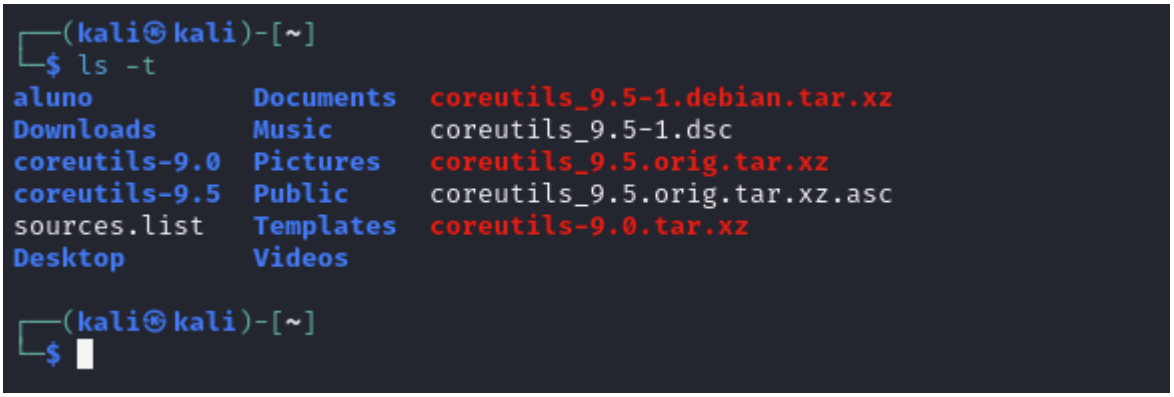
Agatha Christmann de Quadros

O comando **ls** presente nas distribuições Linux tem a função de fazer uma listagem dos arquivos e pastas presentes em um diretório. Por exemplo, no diretório kali de uma distribuição Kali Linux, podemos executar o comando ls e obtemos a seguinte saída no terminal:



Algumas variações do comando podem ser executadas, como por exemplo **ls –size**, que exibe os elementos de um diretório com seus respectivos tamanhos em disco:

Com **ls -t**, podemos exibir a lista organizada por ordem de tempo, exibindo os arquivos/diretórios em uma ordem de mais novo para o mais antigo.



Com o comando **ls -1**, fazemos a lista exibir um elemento por linha:

Existem diversos outros complementos ao comando ls, que podem ser consultados no manual da funcionalidade, que pode ser acessado se o usuário digitar **man ls** no terminal.

Modificações no implementadas no comando ls:

- Modificação das cores:

```
┌──(kali㉿kali)-[~/coreutils-9.0]
└─$ ./src/ls
Total files in the directory: 37
man             doc                     ABOUT-NLS       init.cfg
src             tests                   INSTALL         Makefile.am
gnulib-tests    build-aux               maint.mk        README
lib             m4                      AUTHORS         thanks-gen
config.log      THANKS                  bootstrap       THANKS.in
po              GNUmakefile             bootstrap.conf  THANKStt.in
Makefile        configure               cfg.mk          TODO
config.status   NEWS                    configure.ac
Makefile.in     THANKS-to-translators   COPYING
ChangeLog       aclocal.m4              dist-check.mk
```

Um esquema de cores de padrão azul claro, roxo e branco foi implementado na exibição dos arquivos. Através das seguintes adições no código do arquivo ls.c:

```
// Change the default colors (modification for Purple Linux distribution, by Agatha Quadros)
char *ls_colors = getenv("LS_COLORS");

const char *new_colors = "di=1;96;40:ln=35:so=32:pi=33:ex=1;95;40:bd=34;46:cd=34;43:su=30;41:sg=30;46:tw=30;42:ow=30;43";
setenv("LS_COLORS", new_colors, 1);

if (print_with_color)
  parse_ls_color ();

/* Test print_with_color again, because the call to parse_ls_color
   may have just reset it -- e.g., if LS_COLORS is invalid.  */

if (print_with_color)
  {
    /* Don't use TAB characters in output.  Some terminal
       emulators can't handle the combination of tabs and
       color codes on the same line.  */
    tabsize = 0;
  }
```

Chamamos o environment LS_COLORS, o qual define as cores padrões para o comando. Em seguida, declaramos uma variável *new_colors* que irá conter o novo esquema de cores para LS_COLORS. Cada sigla representa um tipo de arquivo, e cada número, uma cor. Então, passamos esses valores para LS_COLORS através da função *setenv*.

- <u>Exibição do número total de arquivos:</u>

```
┌──(kali㉿kali)-[~/coreutils-9.0]
└─$ ./src/ls
Total files in the directory: 37
man             doc             ABOUT-NLS       init.cfg
src             tests           INSTALL         Makefile.am
gnulib-tests    build-aux       maint.mk        README
lib             m4              AUTHORS         thanks-gen
config.log      THANKS          bootstrap       THANKS.in
po              GNUmakefile     bootstrap.conf  THANKStt.in
Makefile        configure       cfg.mk          TODO
config.status   NEWS            configure.ac
Makefile.in     THANKS-to-translators  COPYING
ChangeLog       aclocal.m4      dist-check.mk
```

Quando o usuário digitar ls, será informado automaticamente sobre o número de arquivos presentes no diretório. Tal funcionalidade foi implementada da seguinte forma:

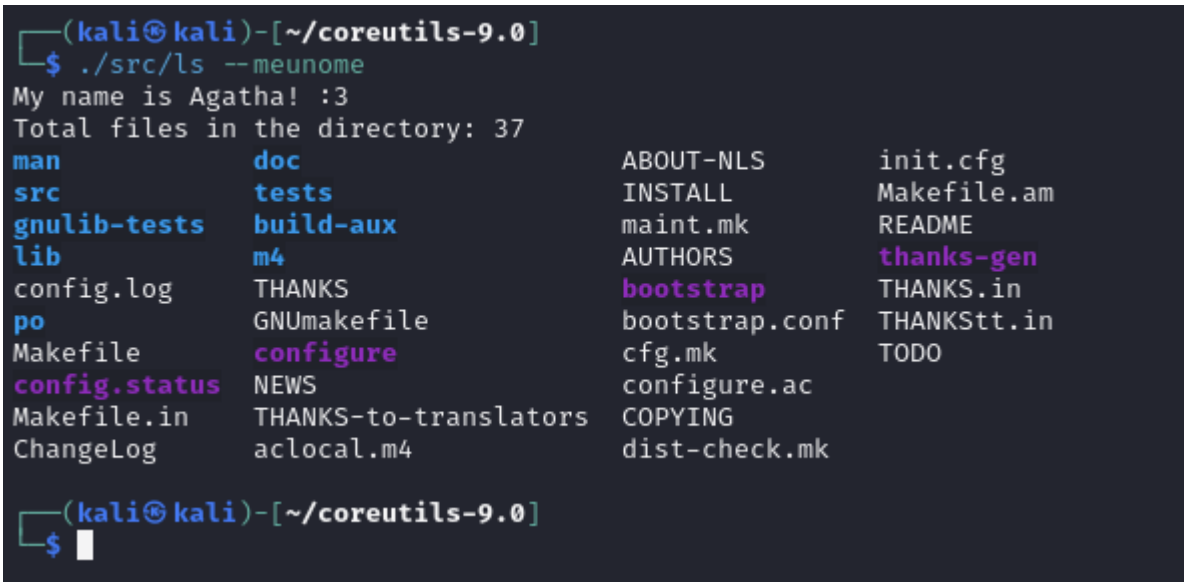
```c
int count_files(const char *dirname){
  struct dirent *entry;
  int file_count = 0;

  DIR *dir = opendir(dirname);

  if(dir == NULL){
    perror("opendir failed");
    return -1;
  }

  while((entry = readdir(dir)) != NULL){
    if (entry -> d_name[0] == '.') //skip hidden files
      continue;
    file_count++;
  }

  closedir(dir);
  return file_count;
}

const char *filesPath = ".";  // Default to current directory

int num_files = count_files(filesPath );
if (num_files >= 0)
  printf("Total files in the directory: %d\n", num_files);
```

O método *count_files* recebe o caminho do diretório (“.” representa o diretório atual), uma variável de diretório é declarada e abre o caminho informado através da função opendir, se a variável dir é nula, um erro é anunciado e o método retorna -1. Se não, o programa itera arquivo por arquivo através de *readdir(dir)*, enquanto ele não retornar NULL. A iteração pula arquivos que começam por “.” considerados ocultos, a leitura do nome é feita através de *d_name[0]*. A cada iteração e arquivo não oculto identificado, a variável

file_count é incrementada. Por fim, o diretório é fechado com closedir e o método retorna a variável *file_count*, que é impressa no *printf* através da variável *num_files*.

- Exibição do meu nome:

  Se **ls –meunome** é digitado, o terminal imprime a seguinte mensagem:



  Primeiro, tivemos que adicionar um atributo de enumerador chamado MEU_NOME:

```
enum
{
  AUTHOR_OPTION = CHAR_MAX + 1,
  BLOCK_SIZE_OPTION,
  COLOR_OPTION,
  DEREFERENCE_COMMAND_LINE_SYMLINK_TO_DIR_OPTION,
  FILE_TYPE_INDICATOR_OPTION,
  FORMAT_OPTION,
  FULL_TIME_OPTION,
  GROUP_DIRECTORIES_FIRST_OPTION,
  HIDE_OPTION,
  HYPERLINK_OPTION,
  INDICATOR_STYLE_OPTION,
  MEU_NOME,
  QUOTING_STYLE_OPTION,
  SHOW_CONTROL_CHARS_OPTION,
  SI_OPTION,
  SORT_OPTION,
  TIME_OPTION,
  TIME_STYLE_OPTION,
  TRANS_RIGHTS,
  ZERO_OPTION,
};
```

  Após, atribuímos o argumento que será utilizado junto com o comando **ls**:

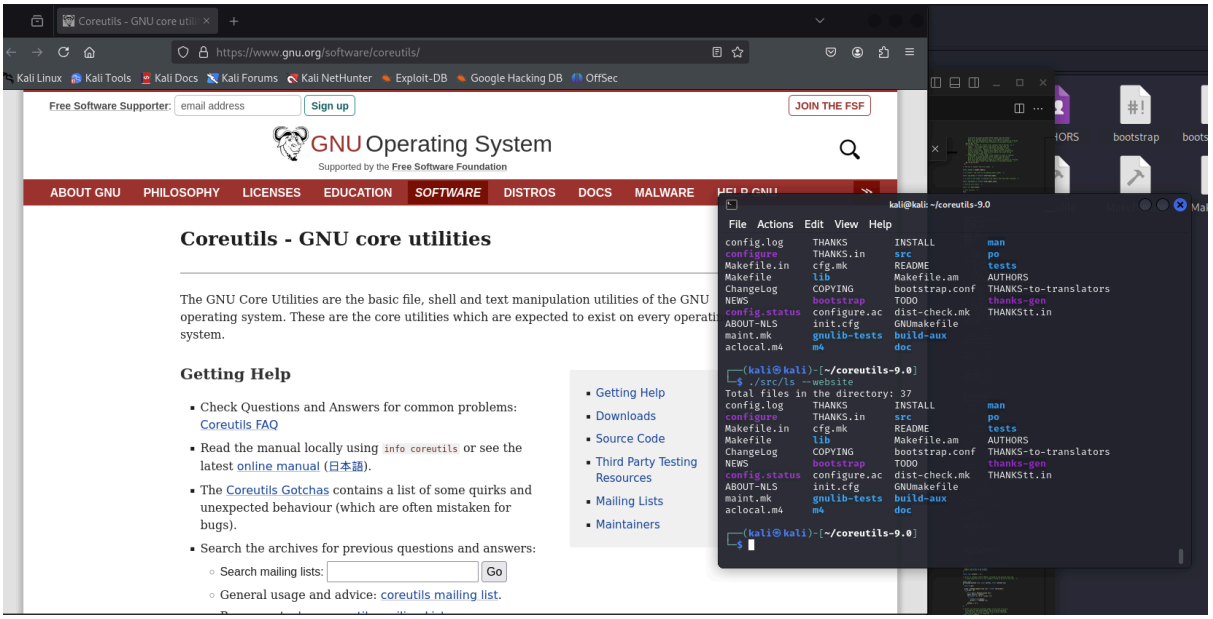
```
{"time-style", required_argument, NULL, TIME_STYLE_OPTION
{"zero", no_argument, NULL, ZERO_OPTION},
{"color", optional_argument, NULL, COLOR_OPTION},
{"hyperlink", optional_argument, NULL, HYPERLINK_OPTION},
{"block-size", required_argument, NULL, BLOCK_SIZE_OPTION
{"context", no_argument, 0, 'Z'},
{"author", no_argument, NULL, AUTHOR_OPTION},
{"meunome", no_argument, NULL, MEU_NOME},
```

Em seguida,adicionamos um case para as opções de argumentos possíveis referenciando o enumerador:

```
case MEU_NOME:
  printf("My name is Agatha! :3 \n");
  break;
```

- *Abrir homepage do GNU Coreutils:*

Ao digitarmos **ls –website**, a homepage do GNU Coreutils é aberta no navegador padrão do sistema:



Para isso, precisamos declarar variáveis no enum de argumentos e definir o argumento website:

```
enum
{
  AUTHOR_OPTION = CHAR_MAX + 1,
  BLOCK_SIZE_OPTION,
  COLOR_OPTION,
  DEREFERENCE_COMMAND_LINE_SYMLINK_TO_DIR_OPTION,
  FILE_TYPE_INDICATOR_OPTION,
  FORMAT_OPTION,
  FULL_TIME_OPTION,
  GROUP_DIRECTORIES_FIRST_OPTION,
  HIDE_OPTION,
  HYPERLINK_OPTION,
  INDICATOR_STYLE_OPTION,
  OPEN_COREUTILS_WEBSITE,
  MEU_NOME,
  QUOTING_STYLE_OPTION,
  SHOW_CONTROL_CHARS_OPTION,
  SI_OPTION,
  SORT_OPTION,
  TIME_OPTION,
  TIME_STYLE_OPTION,
  TRANS_RIGHTS,
  ZERO_OPTION,
};
```

```
  {"website", no_argument, NULL, OPEN_COREUTILS_WEBSITE},
  {GETOPT_HELP_OPTION_DECL},
  {GETOPT_VERSION_OPTION_DECL},
  {NULL, 0, NULL, 0}
};
```

Em seguida, adicionamos um case para o argumento, analisando o retorno da função *system* que irá abrir o navegador com o link. Esse tratamento é necessário pois o sistema dispara um erro quando a possibilidade de um retorno -1 não é considerada utilizando a função *system*. Se o retorno da função não é -1, o seu resultado é executado:

```
case OPEN_COREUTILS_WEBSITE:
int systemRet = system("xdg-open https://www.gnu.org/software/coreutils/");
if(systemRet == -1){
  return -1;
  break;
}
else{
  systemRet;
  break;
}
```