bx021984

Generated by Doxygen 1.8.17

Chapter 1

README

The code is commented entirely and it is clear.
The code is divided logically into smaller files and connected to the main through a header file.
Explained every parameter in the comments for each of my functions.
Listed the external function dependencies for each function (if there is any) inside the comments
There are options to add cost (workload estimate) to a Project, Feature and Folder when creating them both from the wizard and the command line (by adding the optional cost argument at the end e.g –cost=30), as well as adding it later on.
Wizard has a 30 seconds time limit for the input, also when a new input is asked, which characters will be allowed/disallowed in the input is specified custom for that input. Cannot enter empty input or an input with bad character or an input that is longer than the length limit specific for each input.
Renaming feature works on plain folders too, however, renaming a feature will also rename the branch for it as well.

Check List:

- [x] 1. Be able to create basic file structure for project
- [x] 2. Abort if requested project/feature name already exists under 'root' folder. Here 'root' does not mean the / root of the file system, but the folder from which the program is run.
  - [x] 1. Requires checking existing file system for matching name
  - [x] 2. Requires using branching (*if*) to exit program if necessary
- [x] 3. Initialise git repository
  - [x] 1. ***Should*** set up CSGitLab project etc.
- [x] 4. Feature management
  - [x] 1. ***Must*** implement a method of having a shorthand code for feature e.g. F1, F2.1..., stored in a file.
  - [x] 2. ***Must*** implement lookup to facilitate getting path from shorthand code
  - [x] 3. ***Should*** include setting up git branch as appropriate
- [x] 5. Include mechanism for renaming features (subtrees)
- [x] 6. Include mechanism for moving feature to new location in tree (folder hierarchy)
- [x] 7. Output tree diagram - PBS or WBS (svg, using plantuml)
  - [x] 1. Requires tree walk (iterative or recursive)
  - [x] 2. ***Must*** exclude folders that start with a '.'
    ∗∗(note: also excludes bin, lib, tests, src, docs)∗∗
  - [x] 3. ***Should*** use the plantuml tool

    1. ***Could*** implement from scratch (much harder, more marks)

- [x] 8. Time/workload estimate information stored in files in subfolders
  - [x] 1. ***Should*** have mechanisms for adding these from the program not just editing the files
  - [x] 2. ***Should*** include subtrees costs in parent tree total
- [x] 9. Time/workload added to output diagram
  - [x] 1. ***Could*** also produce Gantt chart (using plantuml)
- [x] 10. Output diagram includes links (when used in browser, for example)
  - [x] 1. ***Should*** use plantuml to do this

1. Dependencies information across tree branches
   (a) ***Must*** identify relevant other paths in tree to do this

1.0.0.1 Elite challenges ("Won't do" in MoSCoW terms) (up to 20% for each top level feature)

- [x] 12. Guided work breakdown wizard (Slightly advanced, would require interactive questions/answer handling)
  - [x] 1. Needs a number of sub-features, such as minimum time allocation threshold, user input parsing

1. Multi-user (Advanced, would require some form of permissions model)
   (a) may be best done using a traditional SQL database, but can use flat files. Complex task.
2. Available as web application (Advanced, probably easiest creating a simple embedded server)
   (a) sample code for simple communications between applications will be covered
3. GOAP recommendation of suitable pathway (Advanced, can use existing GOAP library, however)
   (a) GOAP uses a 'heap' data structure

1.0.1 Goals

Identify the key goals you need to complete for the coursework

1. Must have most of the features listed.
1. Final program should be simple to use and user friendly.
1. Should comment everything worthy so it is simple to understand for someone else.
1. Can also have extra features.
1. Should have my features in separate files and link them.

1.0.2 Requirements to fulfill goals

What do you need to be able to meet those goals? This can include clearer specifications, tests (what needs testing, how to test it?), knowledge etc.

1. Plan ahead of what to do.
1. Decide my approach to how to code it.
1. Make the things I am not sure about more clear.
1. Discuss how the final project should be.
1. Make a to-do list and put the things need to be done in order.

1.0.3 Dependencies (mapping goals/requirements etc)

Which requirements relate to which goals - think about drawing out a Product Breakdown for the coursework

1.0.4 Plan:

1. Try doing simple operations on bash and on C to decide which one to use.
1. Design the layout of the program.
1. Investigate how to go about coding for some features including stuff I have not had experience with.
1. Implement in my code the functionalities the program should have.
1. Check if everything is how they are supposed to be and wrap everything up.

1.0.4.1 What were the results of trying things out?

1. I decided to use C for coding.
1. I decided to make modifications on the coding environment to make it easier to work with.
1. I decided to manually test every feature after coding each.
1. I decided to test my code on different environments (operating systems) to see if it works as it should.
1. I decided to use newly learned features of C, after experimenting with them.

1.0.5 List of commands can be used :

Note: Arguments in double square brackets "[[ ]]" are optional.

- 'pm help' for the list of options.
- 'pm' (with no arguments) to run the wizard.
- 'pm create_project [project_name] [[-cost=]]'
  to create a new git project.
- 'pm add_feature [feature_name] [[-cost=]]'
  to add feature.
- 'pm add_tag [tag_name]'
  to add tag.
- 'pm find_tag [tag_name]'
  to find tag.
- 'pm move_by_tag [tag1] [tag2]'
  to move the feature with 'tag1' to the feature with 'tag2'.
- 'pm output_svg [project_name]'
  to create diagrams from the project folder in svg files.
- 'pm rename_feature [old_feature] [new_name]'
  to rename a feature.
- 'pm move_feature [origin_directory] [destination_directory]'
  to move feature.
- 'pm create_folder [folder_name] [[-cost=]]'
  to create a new folder/directory.
- 'pm calculate_workloads [folder_name]'
  to calculate the workload for a project/feature.
- 'pm set_estimate [directory] [workload]'
  to set the workload estimate of a project/feature/folder.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

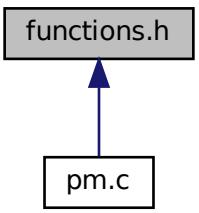
File Documentation

3.1 functions.h File Reference

```
#include "functions/get_input.c"
#include "functions/set_estimate.c"
#include "functions/make_folder.c"
#include "functions/build_structure.c"
#include "functions/git_init.c"
#include "functions/add_feature.c"
#include "functions/add_tag.c"
#include "functions/find_tag.c"
#include "functions/move_by_tag.c"
#include "functions/rename_feature.c"
#include "functions/move_feature.c"
#include "functions/remove_total_workloads.c"
#include "functions/calculate_workloads.c"
#include "functions/fetch_workload.c"
#include "functions/output_svg.c"
#include "functions/wizard.c"
```

Include dependency graph for functions.h:

This graph shows which files directly or indirectly include this file:



Functions

- char * get_input (int size, char *restricted_chars, char *allowed_only_chars)
- void wizard ()
- int make_folder (char *fname, int workload)
- void build_structure (char *pname)
- void set_estimate (char *fname, int workload)
- void git_init (char *pname, int workload)
- void add_feature (char *fname, int workload)
- void add_tag (char *tag)
- char * find_tag (char *tag)
- char * search_tag (char *tag)
- char * check_tag (char *tag)
- void move_by_tag (char *tag1, char *tag2)
- void rename_feature (char *fname1, char *fname2)
- void move_feature (char *fname1, char *fname2)
- void remove_total_workloads (char *pname)
- void calculate_workloads (char *pname)
- void run_calculate_workloads (char *pname)
- int fetch_workload (char *fname)
- int fetch_total_workload (char *fname)
- void output_svg (char *pname)
- int build_wbs (char *pname, char *home_path)
- char * wbs_sub_search (char *wbs, int sub_level)
- char * build_gantt (char *pname, char *gantt)

3.1.1 Function Documentation

3.1.1.1 add_feature()

```
void add_feature (
            char * fname,
            int workload )
```

Adds a feature by creating a folder with the feature name (fname) from the paramter. Also runs 'set_estimate' with (workload) value to set the estimate value for the folder if given. Builds a folder structure inside that folder. Creates and switches to a branch with the name of the feature (fname). Edits or creates a readme.md file, adds it to git and commits it. Switches back to master branch. If the (workload) value is not 0, then set_estimate is called. Aborts if the directory is not inside a git project. External functions used: make_folder, build_structure, set_estimate. Runs 'git status' and the returned value of 0 means the current directory is inside a git repository. Suppresses the output message and the error message of the system call.
Runs make_folder to make a folder for the feature.
Runs build_structure to create the folder structure inside the new feature.
Uses a system call to add a new branch with the name of the new feature.
Uses a system call to switch to the new branch.
Stores a custom text inside readme.md.
Uses a system call to add readme.md to git.
Uses a system call to commit with a custom message.
Suppresses the output message and the error message of the system call.
Uses a system call to switch back to the master branch.
Runs set_estimate if workload value is not 0.

3.1.1.2 add_tag()

```
void add_tag (
            char * tag )
```

Adds the given tag value (tag) to the project/feature/folder by storing it inside the file '.pm_tag'. Aborts if there is already a tag stored. Checks if there is already a tag stored.
If there is no tag stored already, stores the new tag value inside the .pm_tag file.

3.1.1.3 build_gantt()

```
char* build_gantt (
            char * pname,
            char * gantt )
```

3.1.1.4 build_structure()

```
void build_structure (
            char * pname )
```

Used for building the folder structure. Takes the path name (pname) as a parameter to create the folders inside. Uses 'make_folder' to create the folders 'bin', 'docs', 'lib', 'src' and 'tests'. External functions used: make_folder.

3.1.1.5 build_wbs()

```
int build_wbs (
            char * pname,
            char * home_path )
```

3.1.1.6 calculate_workloads()

```
void calculate_workloads (
            char * pname )
```

Calculates the total workload value for the folder with the path/name (pname), and the every subfolder inside it. Stores the total workload value of any folder inside .pm_total_workload file. Takes different approaches depending on what workload values are there for the folder and the subfolder. Aborts if directory with the path/name pname does not exist.
Uses a while loop to iterate through the subfolders and upon finding any, calls itself with the subfolders name recursively to start from the bottom of the tree and go up.

3.1.1.7 check_tag()

```
char* check_tag (
            char * tag )
```

Checks if there is a .pm_tag file, and if there is, if it matches the tag (tag) searched for. Returns the path of the folder the tag is found in, if there is a match. Returns null otherwise.

3.1.1.8 fetch_total_workload()

```
int fetch_total_workload (
            char * fname )
```

Reads and returns the total wrokload value of a folder with the folder name (fname) as an integer. If there is no .pm_total_workload file to read the value from, returns 0.

3.1.1.9 fetch_workload()

```
int fetch_workload (
            char * fname )
```

Reads and returns the wrokload value of a folder with the folder name (fname) as an integer. If there is no .pm_↩ workload file to read the value from, returns 0.

3.1.1.10 find_tag()

```
char* find_tag (
            char * tag )
```

Finds the tag (tag) by searching through the subfolders with 'search_tag' and checking the tag with 'check_tag'. Prints the path of the folder the tag is found in. Also returns the path upon finding a match, which enables the other functions to use this to fetch the path of the folder with the tag. Returns null upon failing. Gets the difference between the home path and the path the tag was found in, and stores it in sole path.
Goes back to the directory the function started running in.
Prints if the tag is not found, and returns null.

3.1.1.11 get_input()

```
char* get_input (
            int size,
            char * restricted_chars,
            char * allowed_only_chars )
```

Sets timer.
Sets the flags for stdin.
Sets stdin to nonblock, so it does not halt the execution for user input.
Takes input until the input is not empty or until the timer runs out. (Input is set to empty when it is too long or has bad characters.)
If 'stdin' file is empty, 'fgetc(stdin)' asks for user input, if it is not empty it reads the first character and removes it from 'stdin'.
Continues until the input is still within the length limit.
Checks for restricted characters only, if there are no allowed characters.
Checks for allowed characters only if there are any. Ignores restricted characters.
If the input has a bad character, notifies the user, asks for a new input and resets the timer.
If the input is longer than the limit specified, notifies the user, asks for a new input and resets timer.
If the input is empty, notifies the user, asks for a new input and resets timer.
If user exceeded the time limit, notifies the user and exits the program.
If user types the exit option (!exit), exits the program.

3.1.1.12 git_init()

```
void git_init (
            char * pname,
            int workload )
```

Attempts to create a folder with the given project name (pname), using 'make_folder', and initializes a git repository on that folder if successfull. Executes command on bash to initialize git. Builds a folder structure using 'build_↩ structure' in that folder. Creates a readme.md file, adds to git and commits. If the workload value (workload) is not 0, stores the workload value inside the project by calling 'set_estimate'. External functions used: set_estimate, make_folder, build_structure. Initialises the git calling git init from the system.
Runs build_structure to create the folder structure inside the new project.
Writes to a new readme.md file.
Adds readme.md to git.
Edits the local config for git to store the name and email values for the user to successfully commit for the first time.
Makes the first commit.
If the workload value is not 0, runs set_estimates with that value.

3.1.1.13 make_folder()

```
int make_folder (
            char * fname,
            int workload )
```

Gets the folder name (fname) as a parameter and creates a folder with that name. Prints wether the operation was successful or it failed (with reasons). If the workload value (workload) is not 0, stores the workload value inside the new folder by calling 'set_estimate'. Returns 1 if successful, 0 if fails. External functions used: set_estimate. Checks the length of the user input. Aborts if it is longer than 255 characters (max length for linux systems).

Checks if there is a space in the folder name. Aborts if there is.
Attempts to create a file and fetches the result. Notifies if succeessful. If it fails, checks the error code and notifies the cause of error for some common cases.

3.1.1.14 move_by_tag()

```
void move_by_tag (
            char * tag1,
            char * tag2 )
```

Moves the folder with the tag tag1 (tag1) to folder with the tag tag2 (tag2). Calls 'find_path' to get the paths for the tags. Aborts if either tag is not found. Uses the system call "mv" to operate. External functions used: find_tag.

3.1.1.15 move_feature()

```
void move_feature (
            char * fname1,
            char * fname2 )
```

Moves a directory (fname1) to another directory (fname2). It creates the parent directories specified in the second path (fname2) argument as 'parent1/parent2/destination', if they do not exist already. Aborts if there is already a directory with the same name in the destination. External functions used: make_folder. Checks if there is a directory to move with the name given as the first argument.
strtok returns the bit of the path name before the first '/'.
Continues to check for the next bit until '/'. And goes inside the directory with that name, creates the directory if it does not exist.
Checks if there is a folder with that name in the destination.
Uses bash command md to move the folders.

3.1.1.16 output_svg()

```
void output_svg (
            char * pname )
```

3.1.1.17 remove_total_workloads()

```
void remove_total_workloads (
            char * pname )
```

Removes the .pm_total_workload files in any directory under the path (pname). This enables 'calculate_workloads' to function properly without the involving the old values. '.pm_workload' files stay the same without any editing.

3.1.1.18 rename_feature()

```
void rename_feature (
            char * fname1,
            char * fname2 )
```

Aborts if there is a bad character
Aborst if the directory is not found.
Aborts if the new name is given in path format.
Series of operations to replace the end bit after the last '/' in the path with the new name.
Uses bash command 'mv' to rename.
If the feature is in a git project and has a branch for it, rename the branch as well.
Checks if there is a branch with the name of the old folder.
Suppresses the output message and the error message of the system call.
Uses system call to run 'git branch -m' to rename the branch.

3.1.1.19 run_calculate_workloads()

```
void run_calculate_workloads (
            char * pname )
```

Runs the 'calculate_wokloads' function, passes it's parameter (pname)to it, and also prints the total workload value for the top folder. Calls 'remove_total_workloads' to get rid of the old total workload values. External functions used: remove_total_workloads. Removes the old total workload values and calculates them again.
Reads the total workload value of the top folder to print it to the user.

3.1.1.20 search_tag()

```
char* search_tag (
            char * tag )
```

Recursively searches through the subfolders for the tag (tag) and calls 'check_tag' to check if there is a match in one of them. Returns the path of the folder the tag is found in, which is returned by 'check_tag', but if null is returned from 'check_tag', it keeps searching. If all the subfolders are searched and tag is not found, returns null.

3.1.1.21 set_estimate()

```
void set_estimate (
            char * fname,
            int workload )
```

Stores the workload value (workload) inside the folder with the name (fname). Aborts if fname is not a directory. Stores the value in the file .pm_workload

3.1.1.22 wbs_sub_search()

```
char* wbs_sub_search (
            char * wbs,
            int sub_level )
```

The recursive function that looks for subfolders and stores the names in the (wbs) variable in wbs style. (sub_level) parameter is passed to record the distance of the folder when function is called recursively. Adds the link for the directories to the diagram. Also lists the workload and the total workload values of a directory after the name of it. Returns the final wbs text in strings. External functions used: fetch_workload, fetch_total_workloads. Skips if the folder name starts with a '.' or is src, lib, docs, tests, bin.
Adds a star for each sub_level before the folder name.

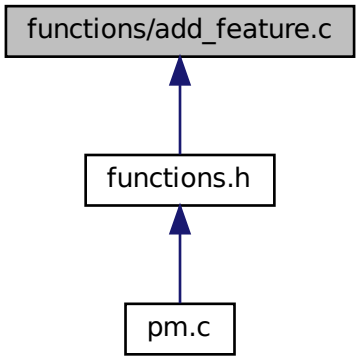
3.1.1.23 wizard()

```
void wizard ( )
```

Wizard that shows options to the user and calls the functions according to the user's choice and input. Uses 'get↩ _input' to get the user input. External function used: 'get_input' and every function for each wizard option. List of possible actions.
Number of new lines to be added.
Add new lines to fill the screen.
Clear terminal
Print options
Switch statement to match the selected option.
Exit
Create Project
Add Feature
Add Tag
Find Tag
Move By Tag
Output SVG
Rename Feature
Move Feature
Calculate Workloads
Set Estimate
Create Folder

3.2 functions/add_feature.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void add_feature (char *fname, int workload)

3.2.1 Function Documentation

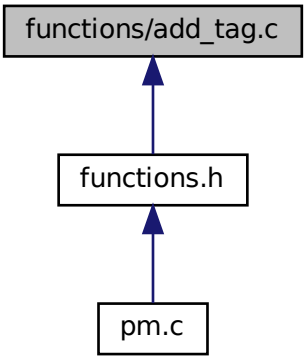
3.2.1.1 add_feature()

```
void add_feature (
            char * fname,
            int workload )
```

Adds a feature by creating a folder with the feature name (fname) from the paramter. Also runs 'set_estimate' with (workload) value to set the estimate value for the folder if given. Builds a folder structure inside that folder. Creates and switches to a branch with the name of the feature (fname). Edits or creates a readme.md file, adds it to git and commits it. Switches back to master branch. If the (workload) value is not 0, then set_estimate is called. Aborts if the directory is not inside a git project. External functions used: make_folder, build_structure, set_estimate. Runs 'git status' and the returned value of 0 means the current directory is inside a git repository. Suppresses the output message and the error message of the system call.
Runs make_folder to make a folder for the feature.
Runs build_structure to create the folder structure inside the new feature.
Uses a system call to add a new branch with the name of the new feature.
Uses a system call to switch to the new branch.
Stores a custom text inside readme.md.
Uses a system call to add readme.md to git.
Uses a system call to commit with a custom message.
Suppresses the output message and the error message of the system call.
Uses a system call to switch back to the master branch.
Runs set_estimate if workload value is not 0.

3.3 functions/add_tag.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void add_tag (char *tag)

3.3.1 Function Documentation

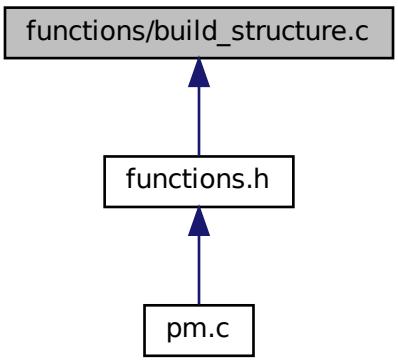
3.3.1.1 add_tag()

```
void add_tag (
            char * tag )
```

Adds the given tag value (tag) to the project/feature/folder by storing it inside the file '.pm_tag'. Aborts if there is already a tag stored. Checks if there is already a tag stored.
If there is no tag stored already, stores the new tag value inside the .pm_tag file.

3.4 functions/build_structure.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void build_structure (char ∗pname)

3.4.1 Function Documentation

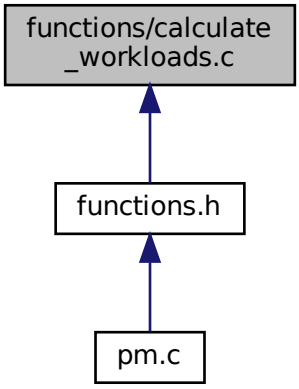
3.4.1.1 build_structure()

```
void build_structure (
            char * pname )
```

Used for building the folder structure. Takes the path name (pname) as a parameter to create the folders inside. Uses 'make_folder' to create the folders 'bin', 'docs', 'lib', 'src' and 'tests'. External functions used: make_folder.

3.5 functions/calculate_workloads.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void calculate_workloads (char ∗pname)
- void run_calculate_workloads (char ∗pname)

3.5.1 Function Documentation

3.5.1.1 calculate_workloads()

```
void calculate_workloads (
            char * pname )
```

Calculates the total workload value for the folder with the path/name (pname), and the every subfolder inside it. Stores the total workload value of any folder inside .pm_total_workload file. Takes different approaches depending on what workload values are there for the folder and the subfolder. Aborts if directory with the path/name pname does not exist.
Uses a while loop to iterate through the subfolders and upon finding any, calls itself with the subfolders name recursively to start from the bottom of the tree and go up.

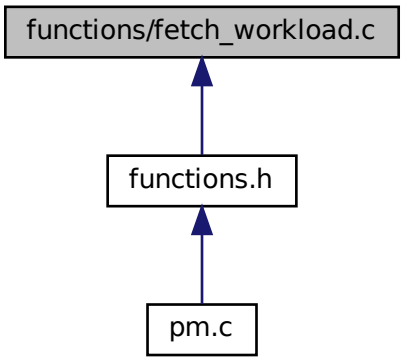
3.5.1.2 run_calculate_workloads()

```
void run_calculate_workloads (
            char * pname )
```

Runs the 'calculate_wokloads' function, passes it's parameter (pname)to it, and also prints the total workload value for the top folder. Calls 'remove_total_workloads' to get rid of the old total workload values. External functions used: remove_total_workloads. Removes the old total workload values and calculates them again.
Reads the total workload value of the top folder to print it to the user.

3.6 functions/fetch_workload.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int fetch_workload (char ∗fname)
- int fetch_total_workload (char ∗fname)

3.6.1 Function Documentation

3.6.1.1 fetch_total_workload()

```
int fetch_total_workload (
            char * fname )
```

Reads and returns the total wrokload value of a folder with the folder name (fname) as an integer. If there is no .pm_total_workload file to read the value from, returns 0.

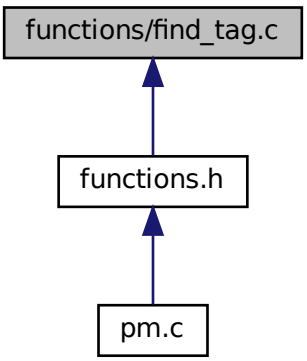
3.6.1.2 fetch_workload()

```
int fetch_workload (
            char * fname )
```

Reads and returns the wrokload value of a folder with the folder name (fname) as an integer. If there is no .pm_↩ workload file to read the value from, returns 0.

3.7 functions/find_tag.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- char * check_tag (char *tag)
- char * search_tag (char *tag)
- char * find_tag (char *tag)

3.7.1 Function Documentation

3.7.1.1 check_tag()

```
char* check_tag (
            char * tag )
```

Checks if there is a .pm_tag file, and if there is, if it matches the tag (tag) searched for. Returns the path of the folder the tag is found in, if there is a match. Returns null otherwise.

3.7.1.2 find_tag()

```
char* find_tag (
            char * tag )
```

Finds the tag (tag) by searching through the subfolders with 'search_tag' and checking the tag with 'check_tag'. Prints the path of the folder the tag is found in. Also returns the path upon finding a match, which enables the other functions to use this to fetch the path of the folder with the tag. Returns null upon failing. Gets the difference between the home path and the path the tag was found in, and stores it in sole path.
Goes back to the directory the function started running in.
Prints if the tag is not found, and returns null.

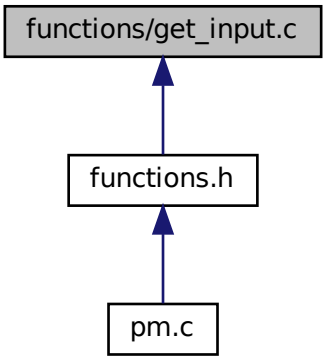
3.7.1.3 search_tag()

```
char* search_tag (
            char * tag )
```

Recursively searches through the subfolders for the tag (tag) and calls 'check_tag' to check if there is a match in one of them. Returns the path of the folder the tag is found in, which is returned by 'check_tag', but if null is returned from 'check_tag', it keeps searching. If all the subfolders are searched and tag is not found, returns null.

3.8 functions/get_input.c File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define INPUT_TIME_LIMIT 30
- #define GOODBYE "\n\033[1;36mExiting the wizard. Goodbye!\033[0m\n\n"

Functions

- void flush_stdin ()
- char * get_input (int size, char *restricted_chars, char *allowed_only_chars)

3.8.1 Macro Definition Documentation

3.8.1.1 GOODBYE

```
#define GOODBYE "\n\033[1;36mExiting the wizard.  Goodbye!\033[0m\n\n"
```

3.8.1.2 INPUT_TIME_LIMIT

```
#define INPUT_TIME_LIMIT 30
```

3.8.2 Function Documentation

3.8.2.1 flush_stdin()

```
void flush_stdin ( )
```

Reads 'stdin' until new line to flush it, so that whatever left in 'stdin' will not be passed when a new input is requested.

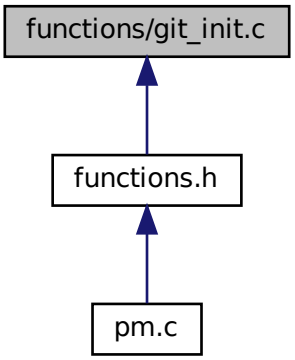
3.8.2.2 get_input()

```
char* get_input (
            int size,
            char * restricted_chars,
            char * allowed_only_chars )
```

Sets timer.
Sets the flags for stdin.
Sets stdin to nonblock, so it does not halt the execution for user input.
Takes input until the input is not empty or until the timer runs out. (Input is set to empty when it is too long or has bad characters.)
If 'stdin' file is empty, 'fgetc(stdin)' asks for user input, if it is not empty it reads the first character and removes it from 'stdin'.
Continues until the input is still within the length limit.
Checks for restricted characters only, if there are no allowed characters.
Checks for allowed characters only if there are any. Ignores restricted characters.
If the input has a bad character, notifies the user, asks for a new input and resets the timer.
If the input is longer than the limit specified, notifies the user, asks for a new input and resets timer.
If the input is empty, notifies the user, asks for a new input and resets timer.
If user exceeded the time limit, notifies the user and exits the program.
If user types the exit option (!exit), exits the program.

3.9 functions/git_init.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void git_init (char *pname, int workload)

3.9.1 Function Documentation

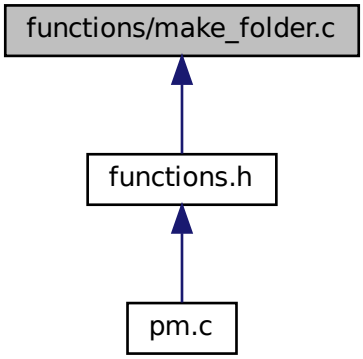
3.9.1.1 git_init()

```
void git_init (
            char * pname,
            int workload )
```

Attempts to create a folder with the given project name (pname), using 'make_folder', and initializes a git repository on that folder if successfull. Executes command on bash to initialize git. Builds a folder structure using 'build_↩ structure' in that folder. Creates a readme.md file, adds to git and commits. If the workload value (workload) is not 0, stores the workload value inside the project by calling 'set_estimate'. External functions used: set_estimate, make_folder, build_structure. Initialises the git calling git init from the system.
Runs build_structure to create the folder structure inside the new project.
Writes to a new readme.md file.
Adds readme.md to git.
Edits the local config for git to store the name and email values for the user to successfully commit for the first time.
Makes the first commit.
If the workload value is not 0, runs set_estimates with that value.

3.10 functions/make_folder.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int mkdir (const char ∗pathname, mode_t mode)
- int make_folder (char ∗fname, int workload)

3.10.1 Function Documentation

3.10.1.1 make_folder()

```
int make_folder (
            char * fname,
            int workload )
```

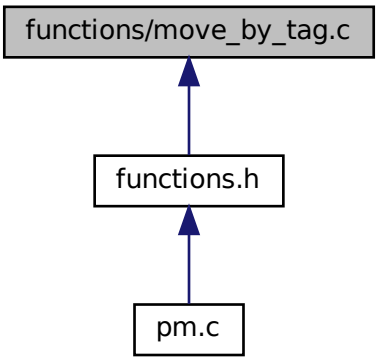
Gets the folder name (fname) as a parameter and creates a folder with that name. Prints wether the operation was successful or it failed (with reasons). If the workload value (workload) is not 0, stores the workload value inside the new folder by calling 'set_estimate'. Returns 1 if successful, 0 if fails. External functions used: set_estimate.
Checks the length of the user input. Aborts if it is longer than 255 characters (max length for linux systems).
Checks if there is a space in the folder name. Aborts if there is.
Attempts to create a file and fetches the result. Notifies if succeessful. If it fails, checks the error code and notifies the cause of error for some common cases.

3.10.1.2 mkdir()

```
int mkdir (
            const char * pathname,
            mode_t mode )
```

3.11 functions/move_by_tag.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void move_by_tag (char *tag1, char *tag2)

3.11.1 Function Documentation

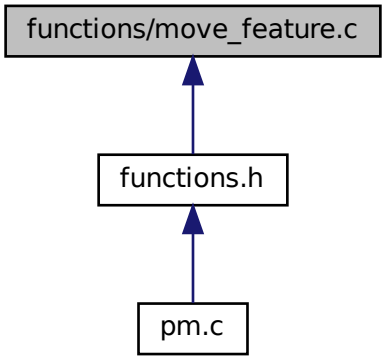
3.11.1.1 move_by_tag()

```
void move_by_tag (
            char * tag1,
            char * tag2 )
```

Moves the folder with the tag tag1 (tag1) to folder with the tag tag2 (tag2). Calls 'find_path' to get the paths for the tags. Aborts if either tag is not found. Uses the system call "mv" to operate. External functions used: find_tag.

3.12 functions/move_feature.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void move_feature (char ∗fname1, char ∗fname2)

3.12.1 Function Documentation

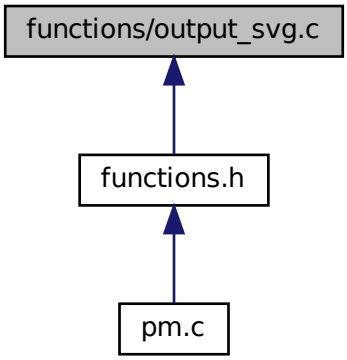
3.12.1.1 move_feature()

```
void move_feature (
            char * fname1,
            char * fname2 )
```

Moves a directory (fname1) to another directory (fname2). It creates the parent directories specified in the second path (fname2) argument as 'parent1/parent2/destination', if they do not exist already. Aborts if there is already a directory with the same name in the destination. External functions used: make_folder. Checks if there is a directory to move with the name given as the first argument.
strtok returns the bit of the path name before the first '/'.
Continues to check for the next bit until '/'. And goes inside the directory with that name, creates the directory if it does not exist.
Checks if there is a folder with that name in the destination.
Uses bash command md to move the folders.

3.13 functions/output_svg.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- char ∗ wbs_sub_search (char ∗wbs, int sub_level)

3.13.1 Function Documentation

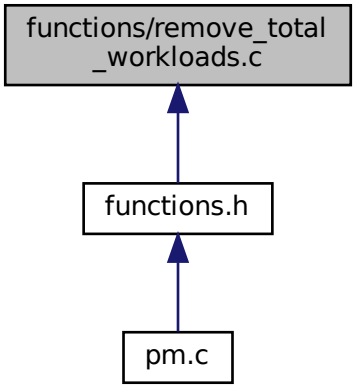
3.13.1.1 wbs_sub_search()

```
char* wbs_sub_search (
            char * wbs,
            int sub_level )
```

The recursive function that looks for subfolders and stores the names in the (wbs) variable in wbs style. (sub_level) parameter is passed to record the distance of the folder when function is called recursively. Adds the link for the directories to the diagram. Also lists the workload and the total workload values of a directory after the name of it. Returns the final wbs text in strings. External functions used: fetch_workload, fetch_total_workloads. Skips if the folder name starts with a '.' or is src, lib, docs, tests, bin.
Adds a star for each sub_level before the folder name.

3.14 functions/remove_total_workloads.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void remove_total_workloads (char ∗pname)

3.14.1 Function Documentation

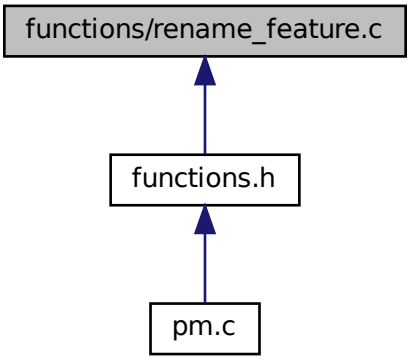
3.14.1.1 remove_total_workloads()

```
void remove_total_workloads (
            char * pname )
```

Removes the .pm_total_workload files in any directory under the path (pname). This enables 'calculate_workloads' to function properly without the involving the old values. '.pm_workload' files stay the same without any editing.

3.15 functions/rename_feature.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void rename_feature (char ∗fname1, char ∗fname2)

3.15.1 Function Documentation

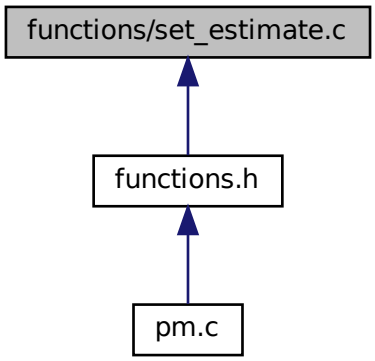
3.15.1.1 rename_feature()

```
void rename_feature (
            char * fname1,
            char * fname2 )
```

Aborts if there is a bad character
Aborst if the directory is not found.
Aborts if the new name is given in path format.
Series of operations to replace the end bit after the last '/' in the path with the new name.
Uses bash command 'mv' to rename.
If the feature is in a git project and has a branch for it, rename the branch as well.
Checks if there is a branch with the name of the old folder.
Suppresses the output message and the error message of the system call.
Uses system call to run 'git branch -m' to rename the branch.

3.16 functions/set_estimate.c File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void set_estimate (char ∗fname, int workload)

3.16.1 Function Documentation

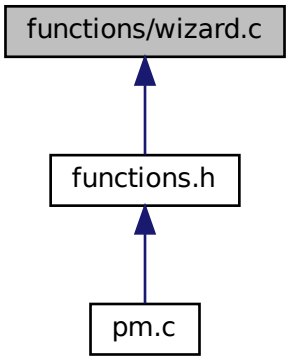
3.16.1.1 set_estimate()

```
void set_estimate (
            char * fname,
            int workload )
```

Stores the workload value (workload) inside the folder with the name (fname). Aborts if fname is not a directory. Stores the value in the file .pm_workload

3.17 functions/wizard.c File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define DIGITS "0123456789"
- #define HEADER " \033[44;33m PROJECT MANAGER \033[0m \n"
- #define GOODBYE "\n\033[1;36mExiting the wizard. Goodbye!\033[0m\n\n"

Functions

- void wizard ()

3.17.1 Macro Definition Documentation

3.17.1.1 DIGITS

```
#define DIGITS "0123456789"
```

3.17.1.2 GOODBYE

```
#define GOODBYE "\n\033[1;36mExiting the wizard.  Goodbye!\033[0m\n\n"
```

3.17.1.3 HEADER

```
#define HEADER " \033[44;33m PROJECT MANAGER \033[0m \n"
```

3.17.2 Function Documentation

3.17.2.1 wizard()

```
void wizard ( )
```

Wizard that shows options to the user and calls the functions according to the user's choice and input. Uses 'get↩ _input' to get the user input. External function used: 'get_input' and every function for each wizard option. List of possible actions.
Number of new lines to be added.
Add new lines to fill the screen.
Clear terminal
Print options
Switch statement to match the selected option.
Exit
Create Project
Add Feature
Add Tag
Find Tag
Move By Tag
Output SVG
Rename Feature
Move Feature
Calculate Workloads
Set Estimate
Create Folder

3.18 pm.c File Reference

```
#include <stdio.h>
#include <sys/types.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <dirent.h>
#include <time.h>
#include <fcntl.h>
#include "functions.h"
```

Include dependency graph for pm.c:

Functions

- int main (int argc, char ∗argv[ ])

3.18.1 Function Documentation

3.18.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Checks if there are any arguments passed from the command line. Tries to match it with the exsiting commands if there is. Returns error message with info if there is a wrong command or a wrong number of arguments entered. RUNS WIZARD IF THERE ARE NO ARGUMENTS PASSED.

Simple 'help' command that gives info about the possible commands and their usage. Used as 'pm help'.
'create_project' command that creates a new git project and builds a folder structure inside. Additional optional argument to set the estimate workload value for it when running. Takes 2 or 3 arguments, aborts otherwise. Used as 'pm create_project project_name [-cost=]'.
'add_feature' command that adds a feature folder and builds a folder structure inside. Additional optional argument to set the estimate workload value for it when running. Takes 2 or 3 arguments, aborts otherwise. Used as 'pm add_feature feature_name [-cost=]'.
'add_tag' command that adds a tag file with the tag inside. Takes only 2 arguments, aborts otherwise. Used as 'pm add_tag tag'
'find_tag' command that finds and prints the path where the given tag exists. Takes only 2 arguments, aborts otherwise. Used as 'pm find_tag tag'
'move_by_tag' command that moves a folder with the first given tag to the folder with the second given tag. Takes only 3 arguments, aborts otherwise. Used as 'pm move_by_tag tag1 tag2'
'output_svg' command that creates a file in plantuml wbs format for the contents of the given project folder and an svg file created from it. Also creates another file in plantuml gantt format with the workload values of thr given project folder and an svg file created from it. Takes only 2 arguments, aborts otherwise. Used as 'pm output_svg project_name'
'rename_feature' command that renames a folder. Can specify the path for the directory in the 2nd argument, as in 'parent/old_folder'. Should give only the new name for the directory in the 3rd argument as in 'new_name'. Takes only 3 arguments, aborts otherwise. Used as 'pm rename_feature old_folder new_name'.
'move_feature' command that moves a directory to another directory. It creates the parent directories specified in the 3rd argument as 'parent1/parent2/destination', if they do not exist already. Takes only 3 arguments, aborts otherwise. Used as 'pm move_feature origin_directory destination_directory'.
'create_folder' command that creates a new folder/directory. Additional optional argument to set the estimate workload value for it when running. Takes 2 or 3 arguments, aborts otherwise. Used as 'pm create_folder folder_name [-cost=]'.
'run_calculate_workloads' command that calculates the workload value of a folder with path/name 'pname', and the every subfolder inside it. Stores the workload value of each folder and subfolder to a file called '.pm_workload' inside it. Takes only 2 arguments, aborts otherwise. Used as 'pm calculate_workloads folder_name'
'set_estimate' command that stores the given workload estimate value inside the given directory. It creates/writes the value to a file with the name ".pm_workload". Takes only 3 arguments, aborts otherwise. Used as 'pm set_↩ estimate directory workload'.

3.19 README.md File Reference

Index