====== pypc =====

===== document =====

Di SONG

songdi19@gmail.com

content

Features

Basically, The pypc can do a preprocess work for any text files, such as code and plaint text, whatever. of course, it is more valuable for processing the code files. And, it supports boolean, integer, float and string data types. and supports syntax check as well.

The specification of the files the pypc

The file should have a character or a string for a single line comment. such as "//" for java, "#" for python.

What it can do

Simple example of a preprocess statement

```
//     #define BOOL_VALUE  True
//     #define INT_VALUE  123
//     #ifdef BOOL_VALUE
        something here when BOOL_VALUE is TRUE
//      #ifdef INT_VALUE == 123
        sub if block:
            // #<< INT_VALUE
//      #else
            INT_VALUE is NOT 123
//      #endif
//      #else
        BOOL_VALUE is NOT TRUE
//      #endif
```

After preprocessing, we can get the code below:

```
        something here with BOOL_VALUE is TRUE
                sub if block:
            // INT_ VALUE == 123
```

How to run

Command Line:

```
python pypc.py -s srcfile [-d destdir [-e [-i initfile [-m comment ]]]]
```

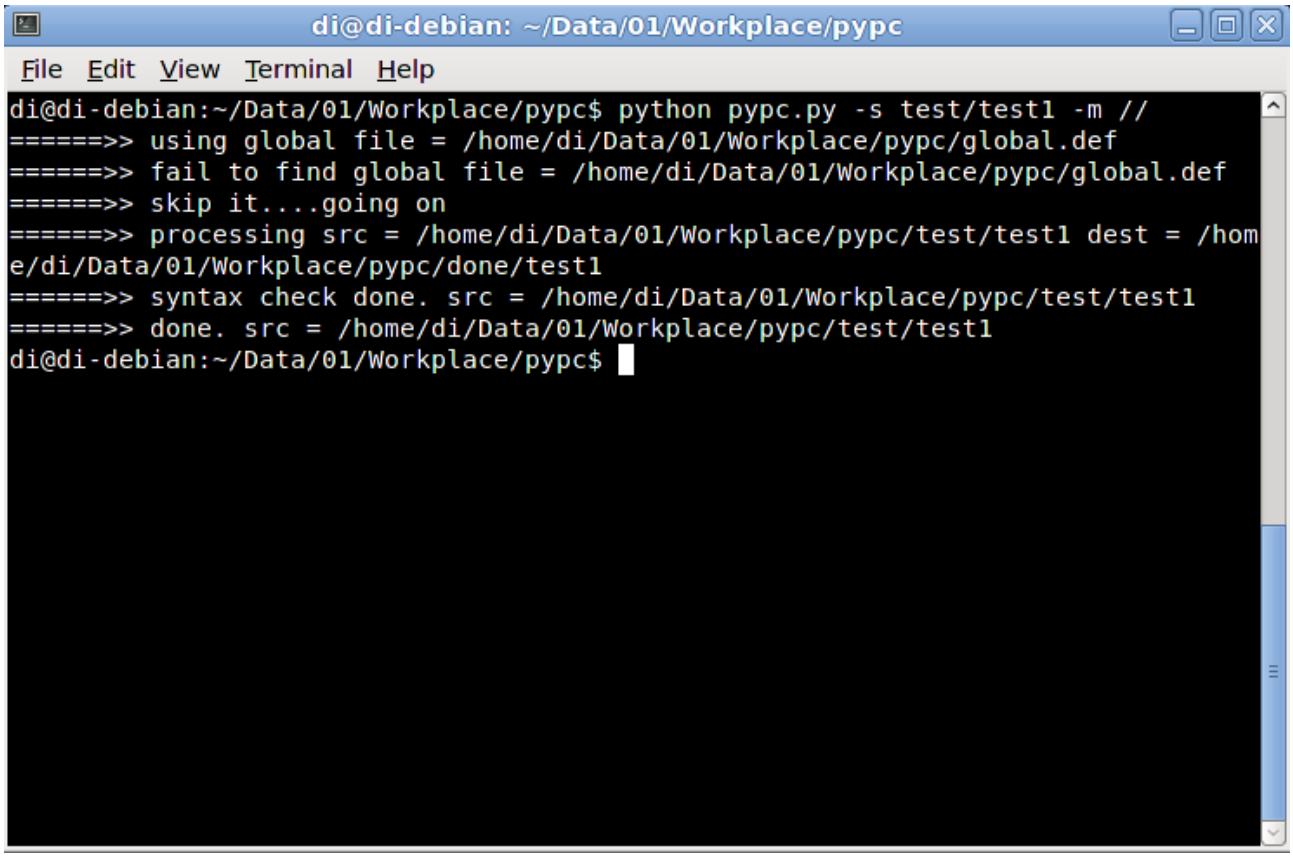
-s Source file or directory.

-d Destination file or directory.

-e flag for export, setting -e to export a code version with the parameters you set. Or just comment the useless code, which is easy to debug your code, because the line number of code file will not be changed after preprocessing.

-i Define a initial file, this file will be loaded firstly. The default name of init file is "global.def". You can define some global variables in this file.

-m Define yourself mark for comment. The default is "#".



Brother project

If you like to use these futures with Java, please see my another project.

per-processor-java (http://code.google.com/p/pre-processor-java/)

A sample test sample

For any text files, such as code and plaint text, they maybe like below: (we use python's comment '#')

```
# #define my_string "Hello pypc"
# #ifdef my_string
        if block: your codes or something
# #<< my_string
# #else
     else block: your codes or somthing
# #endif
```

save it as demo.txt. Let us preprocess it with the pypc. In shell, we input:

```
python pypc.py -s demo.txt
```

You can see:

```
di@di-debian:~/Data/01/Workplace/pypc$ python pypc.py -s demo.txt
=======>> using global file = /home/di/Data/01/Workplace/pypc/global.def
=======>> fail to find global file = /home/di/Data/01/Workplace/pypc/global.def
=======>> skip it....going on
=======>> processing src = /home/di/Data/01/Workplace/pypc/demo.txt dest = /home/
di/Data/01/Workplace/pypc/done/demo.txt
=======>> syntax check done. src = /home/di/Data/01/Workplace/pypc/demo.txt
=======>> done. src = /home/di/Data/01/Workplace/pypc/demo.txt
di@di-debian:~/Data/01/Workplace/pypc$
```

And the preprocessed file "demo.txt" is in the "done" directory which is in your current path.

```
# #define my_string "Hello pypc"
# #ifdef my_string
        if block: your codes or something
# my_string == Hello pypc
# #else
```

```
   #      else block: your codes or somthing
# #endif
```

Actually, only "if block: your codes or something" is available.

How to write a preprocess script

Attention: Any statements must be written in a independent line.

Define a local varible

- **Define a boolean variable**

  Syntax:

  comment #define PARAM **TRUE|True|true|FALSE|False|false**

  Example:

  Java: // #define bool true

  Python: # #define debug false

- **Define a integer variable**

  Syntax:

  comment #define PARAM i**nteger_number**

  Example:

  Java: // #define num 123

  Python: # #define size -256

- **Define a float variable**

  Syntax:

  comment #define PARAM **float_number**

  Example:

  Java: // #define num 123.05

  Python: # #define data -23.4

- **Define a string variable**

  Syntax:

  comment #define PARAM **“string”**

  Example:

Java: // #define str “Hello pypc”
Python: # #define str “This is a string”

Define a global variable

You can access a global variable in any files during one processing procedure. The global variable should be defined in a initial file. See detail in “Use a initial file”.Using comment #define global PARAM value to define a global boolean,integer,float and string variable.

Example:

```
// #define global gloabl_bool True
# #define global global_int 20
// #define global global_float -33.3
# #define global global_name "Di SONG"
```

Tip: *When a local variable has the same name with a global variable. For this problem, the preprocessor will search this variable in the local namespace firstly, if not find, then search it in the global namespace. For avoiding this situation, you should add a prefix before a global variable. Such as global_PARAM. Or using global reference, see “****using global variable****” below.*

Use a initial file (global.def)

Before processing the source files you want, a initial file will be loaded at the beginning. If you do not declare yourself initial file, one default file “global.def” will be loaded automatically which is in the current directory. If the “global.def” does not exist. The preprocessor will skip the initial file and go on processing the source files.

You only should define your global variables in the initial file. Here is an example:

```
/* example of global.def */
// #define global global_bool False
// #define global global_int 123
// #define global global_float 100.0
// #define global global_str “one string”
```

#include “filename”

One main function of this statement is to orginase a pre-processing plan. You can write a processing plan in one file with “#include” statement which includes some source files you want to do pre-process. This statement can be written in any source files and a file's anywhere.

Example:

```
/* a pre-processing plan */
# #define global plan_name “plan demo”
# #define plan_1 True
# #ifdef plan_1
    # #include “my_test1.txt”
    # #include “my_test2.txt”
# #else
    # #include “my_test3.txt”
# #endif
```

if-else statement

Syntax:

comment #ifdef express

If the express is true, processing “if” block

[comment #else]

Otherwise processing “else” block

comment #endif

Example:

Java: **// #ifdef a == 1**

if block:

// #else

else block:

// #endif

Python: **# #ifdef a == 1**

if block:

 # #else

else block:
 # #endif

if-not-else statement

Syntax:
comment #ifndef express
If the express is false, processing “if” block
[comment #else]
Otherwise processing “else” block
comment #endif

Example:
Java: **// #ifndef a == 1**
if block:
// #else
else block:
// #endif

Python: **# #ifndef a == 1**
if block:
 # #else
else block:
 # #endif

Express

- **Boolean**
  Synatx:
  comment #ifdef|#ifndef PARAM **==|!= TRUE|True|true| FALSE| False|false**
  Example:
  Java: **// #ifdef** bool **== true**
  Python: **# #ifndef** bool **!= False**
- **Integer**
  Syntax:
  comment #ifdef|#ifndef PARAM **==|!=|>|>=|<|<=** integer_number

Example:

Java: **// #ifdef** int **==** 100

Python: **# #ifndef** num **<=** -25

- **Float**

Syntax:

comment #ifdef|#ifndef PARAM **==|!=|>|>=|<|<=** float_number

Example:

Java: **// #ifdef** rate **!=** 0.25

Python: **# #ifndef** version **>=** 1.01

- **String**

Syntax:

comment #ifdef|#ifndef PARAM **==|!=|>|>=|<|<=** “one string”

return a result of the compare between both with ACSII

Example:

Java: **// #ifdef** str **!=** ”Hello”

Python: **# #ifndef** ver **>=** ”1.01 bate”

- **Value**

Syntax:

comment #ifdef|#ifndef PARAM1 **==|!=|>|>=|<|<=** PARAM2

return a result of the compare between the values of both sides

Example:

Java: **// #ifdef** str1 **!=** str2

Python: **# #ifndef** num1 **>=** num2

Single item express

Syntax:

comment #ifdef|#ifndef PARAM

When the PARAM is boolean, if and only if PARAM exists and is true, the result of the express is true. When the PARAM is NOT boolean, if and only if PARAM exists, the result of the express is true.

#<< variable

It can output the value of this variable.

Syntax:

comment #<< variable

Example:

Java: **// #<<** str

Python: **# #<<** version

Using global variable

If you wan to refer a global variable directly, that is very easy. Only need to add a “global” in front of parameter, so the value of this parameter is from global namespace.

Example:

```
# #ifdef global var == “string”
# #ifndef global var != “string”
# #<< global var
```

For the “value” express of ifdef/ifndef, ONLY can add “global” for the first parameter. I take into account that it is useless to compare two variables that both are in the global namespace.

Such as: `# #ifdef global var1 == var2`