Yocto Build Task Process

Today I understand the build process in yocto project how bitbake build any recipes and how many process in run background to build this recipes.
For example :- I have created one layer(meta-mylayer) in poky folder I have added one recipes name in this folder name is helloworld.
First create proper recipes file in this folder and create local file location for this file.
Check how many process and task can do for build this process in background by using bitbake.

Check this process in build folder.
$ bitbake -c listtasks recipe-name

For example :- $ bitbake -c listtasks helloworld
Show all the build tasks are :-
do_fetch
do_unpack
do_patch
do_configure
do_compile
do_install

1. do_fetch

This task is used to fetch the source data through use of fetcher and fetcher may be like git, https, http , local file(file).
After fetch is successful.
Saved in downloads folder for your recipes soure data.
Also saved in working directory folder **/tmp/work/core2----/hello-world/1.0-r0/temp**
If you run only this tasks then it create only one folder in work directory name is temp means temporary data and not show actual recipes data.

Execute the do_fetch task only for this recipes and run this command in build folder.
$ bitbake -c do_fetch helloworld

Check your file is present or not in your download folder.
$ ls | grep helloworld

2.do_unpack

This task is used to unpack the download source data in work directory folder.
After do_unpack is successful then it show all recipes data in work directory folder or otherwise it show only temp folder if do_unpack is not used or not successful.

Execute the do_unpack task for this recipes and run this command in build folder.
$ bitbake -c do_unpack helloworld

Check the output for this recipes by using do_unpack task in workdirectroy.
$ bitbake -e helloworld | grep ^WORKDIR=

3. do_patch

This task is used to applies patches in your building recipes.
It locates or applies patches in your building recipes.
If you have not apply patches for your recipes then you will skip this this task.

Define this patch file in your recipes file(.bb) file and use Source variable like.
SRC_URI:append = " file://001-patch-example.patch"   << this is patch file >

If you skip this task then do not show patches folder in your recipe folder in the working directory.
If apply patches then it show patches folder in your recipe folder in the working directory.

Execute only the patches file for a specific recipes.
$ bitbake -c do_patch helloworld

Enter your working directory
$ bitbake -e helloworld | grep ^WORKDIR=

4. do_configure

It enabling or disabling the specific feature in your recipes to define it for specific purpose.
syntax :-
this task is used in .bb file
do_configure() {
}

For example I have write header files for some recipes or normal c program file which is used this header files.

```
do_configure() {
            echo  " #define WELCOME y" > helloworld.h
            echo  " #define HELLOWORLD y" >> helloworld.h
}
```

> means write.
>> means append.

Execute only the do_configure task for this recipes.
$ bitbake -c do_configure helloworld

note :: show the changes in your header files. Show data HELLOWORLD in this helloworld.h file.

5. do_compile

This task is used to compile or compilation of your files and generated output for your file in binary form.
Norma file convert Binary or object file.
syntax :-

```
do_compile() {
}
```

important terms used in compilation terms.
CC = cross compiler
CFLAGS = cross compiler flags which is applicable in your binary file.
LDFLAGS = linker or libraries flags which is applicable in your binary file.
S = source your files.
D = destination directory.
m = mode of ownership.
d = creates a destination directory.
bindir = creates output in /usr /bin.
WORKDIR = working directory.

Check the flags output for the specific parameters used in do_compile task.
first check CC

```
$bitbake  -e helloworld | grep CC=
```

Check CFLAGS

```
$bitbake  -e helloworld | grep CFLAGS=
```

Check LDFLAGS

```
$bitbake  -e helloworld | grep LDFLAGS=
```

Note :: If do_compile task is not run then not create binary output or not show binary output in the working directory for your recipes only show data not show compile binary output. So it is mandatory for this generated binary output.

Structure of do_compile task

```
do_compile() {
              ${CC}   ${CFLAGS}    ${LDFLAGS}    ${S}/helloworld.c  -o hello
}
```

helloworld.c is a actual original file of your c program.
hello is a binary or object file for your c program.

Execute the do_compile task manually .

```
$bitbake  -c do_compile helloworld
```

Check the binary output in the working directory.

```
$bitbake  -e helloworld | grep ^S=                    >> This is show source file
$bitbake  -e helloworld | grep ^WORKDIR=              >> This is show where is source file is created
```

Note :: Compile successfully then show your binary output in work directory for your recipes and this binary is not run your build host machine because this is arm based binary output. It uses arm based flags to compile this output.

6. do_install

This task will install the build package in your recipes make a part of your build rootfs file which is run in your embedded board.
If you not run this task then it not show image type folder in your recipes work directory folder and not show where is installed your final output for this packages which is written your recipes.
It show **image** folder name in your recipes working directory.

Syntax :-

```
do_install() {
}
```

I have used bindir means I have created my binary in /usr /bin / my binary output.

Proper syntax :-

```
do_install() {
                    install  -d ${D}${bindir}
                    install  -m 0755 ${S}/hello      ${D}${bindir}
}
```

0755 means :-
0 means it is regular file.
7 means it is read , write and execute file .
5 means read and execute.

Execute the do_install task :-

```
$bitbake  -c do_install helloworld
```

Check output for your working directory in the image folder name.

```
$bitbake  -e helloworld | grep ^D=
                 or
$bitbake  -e helloworld | grep ^WORKDIR=
```

and check manually

/tmp / work / core2------ / helloworld / 1.2.r0 / image / usr / bin / hello

hello is your final binary output for your helloworld.c program.

** END **