Building bash-completion for clang

Yuka Takahashi

April 3 2017

1. Abstract

Shell completion is a function which developers use everyday. Typing "*ls -*" and pressing *[tab]* will return a list of probable options, and typing "*sudo apt ins*" and pressing *[tab]* will complete the last argument (in this case "install"). Since each command takes different arguments, shells need to be taught how to complete arguments for each command.

The aim of this proposal is twofold:

- There's no bash autocompletion support for clang at the moment. Therefore I will build the bash-completion which **works not only for the current version but for all future versions of clang.**
- Not merely implement completion for bash, but also **make this project highly portable to any other shells (zsh, fish..etc) by implementing completion behavior in clang internals.**

2. Introduction and Goals

The goal of this project is to **provide powerful and generic bash completion for all command-line clang users.** For example, the following completions will be provided:

- "*clang [tab]*" will list all flags and their descriptions
- "*clang -fno-st[tab]*" will return:
  -fno-stack-protector   Disable the use of stack protectors
  -fno-standalone-debug Limit debug information produced to reduce size of debug binary
- "*clang -std=[tab]*" will list all available C++ versions
- "*clang [tab]*" will list all source files in current directory
- "*clang -fplugin=[tab]*" will list all .so files in current directory
- "*clang -isysroot [tab]*" will list all directories under current directory

- "*clang -analyzer-checker=unix.[tab]*" will return:
  unix.API unix.Malloc unix.MallocSizeof
  unix.MismatchedDeallocator unix.StdCLibraryFunctions
  unix.Vfork unix.cstring.BadSizeArg unix.cstring.NullArg
- "*clang -fsanitizer=le[tab]*" will return "*-fsanitizer=leak*"
- "*clang -stdlib=[tab]*" will return all available C++ standard library
- "*clang -mllvm=[tab]*" will return:
  -fla control flow flattening pass -sub instruction substitution pass
  -bcf bogus control flow pass

note: *[tab]* means pressing tab here, or any key invoking an autocompletion functionality.

Also, it will be possible to switch the value when *[tab]* is pressed again and again, like many shell completions. E.g. when a command is "*clang -std=*" and the user pushes the tab once, the completion will be c++, and when tab is pressed again, the completion will be c++11.

3. Background

3.1 Idea for the implementation

At first, I had three implementation ideas.

- Hardcode directly to bash-completion system
  This would have been the easiest implementation, but it was not transferable to other shells, and also I had to modify it every time new flags were added to clang.
- Mostly dependent on bash-completion, but build a flag in clang which returns specific information. (Eg. building flag returns supported C++ version for -std=)
  This is more transferable than the previous one, and also, I don't have to modify it as newer versions of clang are released. However, it is not realistic to make flags for each 4.1.3 flag (Completion for specific flags), so it will have defective implementation.
- Building -autocompletion flag in clang internal
  This would be the most transferable and the simplest for users.

3.2 Present situation of shell-completion for clang

In fact, there is already completion for clang in zsh [1]. However, this completion is not efficient because it uses the same code as gcc. Also, all code is implemented on zsh-completion itself, so it does not know about clang internals. It is impossible to effectively use the future clang version and has low portability.

3.3 The reason for using bash

In the future, this completion system will be ported to other shells (zsh, csh...etc). Transporting will not be very difficult because the main behavior is implemented inside clang. However, I thought that implementing for bash is the best choice for the first implementation because bash is the default shell found on most systems and is widely used.

4. Implementation plan

In this project, the command-line flag named "-autocompletion" will be added to clang, and bash will handle this flag for clang autocompletion. When the user presses *[tab]* in bash, bash will execute "*clang -whatsoever[tab] -autocompletion*". Executed clang driver will search for the appropriate flags/files/values and return them with descriptions. Bash will receive the return value and print the completion result and description for "*whatsoever*". This approach increases maintainability and reduces the cost of transporting to other shells compared to directly implementing to bash-completion.

Implementation can be divided into two parts:

1. **clang part: implement -autocompletion flag in clang**
2. **bash part: write bash code to have clang autocomplete and merged it into bash-completion**

These are described in more detailed below.

4.1. Clang part: Implementation of the –autocompletion flag

Behavior of –autocompletion:
When clang receives the –autocompletion flag, it will autocomplete flag just before –autocompletion. (Eg. when the command is "*clang -fno-stac[tab]*", "*-fno-stac*" is target). This specification helps when completing a command, such as "*clang -fno-stac[tab] test.cpp*". It will complete the flag just before tab, so it returns "*clang -fno-stack-protector test.cpp*".

I will implement the –autocomplete flag in the clang driver and add missing information (acceptable file types, available values, descriptions of values, etc.) to clang/include/clang/Driver/Options.td or somewhere else in order to enable clang driver to handle flags collectively.

Completion of flags can be divided into three types.

1. **Completion of flag itself**
   For example, "*clang -hel[tab]*" is in this category. Implementation of this category requires searching for available flags among all clang flags, and listing them with descriptions.
2. **Completion for files and directories**
   "*clang -S [tab]*" is in this category. Implementation of this category requires searching the appropriate files, directories, and returns. Eg. "*clang -fplugin=[tab]*" is required to return only .so extension files.
3. **Completion for specific flags**
   For example, "*clang -std=[tab]*" and "*clang -analyzer-checker=[tab]*" is in this category. **This category requires the most effort and time**, because different implementations or definitions are necessary for every flag in this category.

4.2. Bash part: Implementation for bash

Implement clang autocompletion patches for bash-completion [2] and send pull-requests to them. Bash will specifically handle *[tab]* when arg0 is "clang". When tab is pressed and arg0 is "clang", bash will fork the process and exec "*clang -(whatsoever flag) -autocomplete*". Clang will return completed string or available completion list, so bash will print it. Execution speed must be fast enough for practical use. I have already made contact with bash-completion developer Ville about this project.

5. Timeline

Clang part (4.1) will be implemented between week 1 to week 9. Bash part (4.2) will be implemented between week 10 to week 12. Week 13 to 16 will be used to write documents and fix bugs.

Week:

1. **May 5 – May 11**
   Change Options.td for -autocompletion. This is necessary for the clang parser to handle -autocompletion flag. Also, start reading Driver.cpp and implement outlines.
2. **May 12 – May 18**
   Complete implementing outlines in Driver.cpp. Define functions and necessary classes. Add missing information of 1st type flag (4.1.1 Completion of flag itself) to Options.td.
3. **May 19 – May 25**
   Complete 1st type of flag completion (4.1.1 Completion of flag itself). Search flags by forward match, and complete it if available flag is only one, or list them with descriptions if there are many. Pressing *[tab]* to flip flags will be also available in this period.
4. **May 26 – June 1**
   Implement 2nd type flag (4.1.2 Completion for files and directories) handling in Driver.cpp. Add missing information of 2nd flag type to Options.td.
5. **June 2 – June 8**
   Complete 2nd type of flag completion (4.1.2 Completion for files and directories). Appropriate file types for each flags in 4.1.2 group must be specified to Options.td. According to this information, clang driver will search under directories and print files/directories and complete/list them. Pressing *[tab]* to flip files/directories will be also available during this period.
6. **June 9 – June 15**

Start coding 3rd type of flag completion. (4.1.3 Completion for specific flags). There are approximately 127 flags in this group (this is the number of flags which require "value" for second argument), and information (available values, descriptions of values, etc.) in Options.td for this group is widely missing. Therefore, I will have to spend a large amount of time for this group. First week (this week) and second week will be used to add information to Options.td. Third week and fourth week will be used to implement clang driver to handle this information.

7. **June 16 – June 22**
   Complete implementation for Options.td. Make sure that each 3rd type flag has its description, available values, and descriptions of values.
8. **June 23 – June 29**
   Start implementing Driver.cpp for 3rd flag type. Try to handle flags collectively, so that when someone adds a new flag to clang in the future, they don't have to change Driver.cpp for completion, but only adding information of the new flag to Options.td will work.
9. **June 30 – July 6**
   Complete 3rd type flag implementation this week. Writing tests and fixing bugs. Pushing *[tab]* to flip values will be also available by this period. It is required to satisfy all 4.1 behaviour by this week.
10. **July 7 – July 13**
    Start implementation for bash (4.2). I've taken advice from bash-completion developer Ville to refer to existing bash-completion codes and programmable completion section of the bash man page. Also, make a clang completion file and change the Makefile in the bash-completion repository.
11. **July 14 – July 20**
    Implementing main part of bash implementation. Handle *[tab]* if arg0 is "clang", and fork and exec "*clang -whatsoever -autocompletion*",and print the return value from clang. In this implementation, I will have to make sure that completion speed is fast enough for practical use.
12. **July 21 – July 27**
    Complete bash implementation for -autocomplete. Make sure all features at 4.2 are implemented.
13. **July 28 – August 3**
    Testing for both 4.1 and 4.2. Make sure that the project works coherently.
14. **August 4 – August 10**
    Fix bugs which emerged during testing.

15. **August 11 – August 17**
    Write detailed documentation about this program.
16. **August 17 – August 22**
    Create final report for final review.

6. Personal Information

- Name: Yuka Takahashi
- Residence: Japan
- Email: yukatkh@gmail.com
- Affiliation: CS student at the University of Tokyo
- Timezone: UTC+9
- IRC nick: yamaguchi/yamaguchi1024
- CV
- Website

6.1 Availability

I will be fully available during the GSoC period and able to spend over 40 hours per week because I have almost no courses this year. I'm willing to spend a great deal amount of time on this project.

6.2 Previous experiences

I have experience in building shell [3], so I am not concerned about the second part of the implementation.

Regarding the 1st part of implementation, until recently, I didn't have commits for LLVM/clang. However, thanks to advice from potential mentor, Raphael, I had been working hard to fix those bugs [4][5][6]. These patches are now reviewed in [7][8].
This was a bug where some options were not working in Unix, so, I think this experience will help implementing –autocompletion flag.

6.3 Motivation

I enjoy participating in CTF (Capture The Flag, which is collective term for security contest), and I am very interested in the field of computer security.

When I participated in security camp (a camp to grow white-hackers, sponsored by the Japanese government) last year, I was fascinated by system software and hardware architecture, including assembly, compiler, and CPU.
I was studying about them by reading books and blogs on my own, but I wish to contribute to the OSS community.

When I discovered this project, I was certain it was for me, because it requires knowledge about shell, and also it makes me very familiar with LLVM/clang internals and flags.

7. References

[1]
https://github.com/zsh-users/zsh/blob/master/Completion/Unix/Command/_gcc
[2] https://github.com/scop/bash-completion
[3] https://github.com/yamaguchi1024/shell
[4] https://bugs.llvm.org//show_bug.cgi?id=11503
[5] https://bugs.llvm.org//show_bug.cgi?id=26834
[6] http://bugs.llvm.org/show_bug.cgi?id=32280
[7] https://reviews.llvm.org/D31495
[8] https://reviews.llvm.org/D31591