How to Add a New Subcommand

The subcommand framework is responsible for maintaining available subcommands in GEAR. It encapsulates and abstracts the basic routines of finding, parsing, executing and printing help information of the subcommand specified by user. It organizes module (subcommand) specific arguments into different argument classes to prevent them from messing up in a single big class (just like the old big damn CmdArgs), without bothering each subcommand to redundantly implement the same (or similar) command-line parsing, execution and help-formatting routine.

The framework consists of three abstract classes at the top level in package gear: CommandArguments, CommandImpl, and Command. You need to extend these three classes if you want to add a new subcommand into the subcommand framework of GEAR.

## ****Define and Implement a CommandArguments Subclass****

CommandArguments and its subclasses are designed to be plain data classes and only contain getters and setters. All the arguments of a specific subcommand must be packed into a CommandArguments subclass, in which the arguments can be of any type.

## Define and Implement a CommandImpl Subclass

A CommandImpl subclass implements the detailed routine (usually statistical method) of a subcommand. The only thing all CommandImpl subclasses must do is to override the execute method. This is the entrance method of a CommandImpl subclass. A Command instance calls this method on its associated CommandImpl partner to execute the subcommand, passing in the CommandArguments instance it just parsed.

## Define and Implement a Command Subclass

A Command subclass connects the corresponding CommandArguments subclass and CommandImpl subclass together to make a subcommand work. It prepares and parses the command-line options, converts and packs the option values into the corresponding CommandArguments instance, and then executes the subcommand by calling the execute method on the corresponding CommandImpl instance.

### Override getName method

### Add subcommand aliases by calling the inherited addAlias method in the constructor (optional)

### Override the getDescription method

The returned value will be used in the subcommand list in ‘gear help’ (top level help listing all the available subcommands) to shortly describe the command. Generally speaking, it should not be longer than 50 characters.

### Override the getLongDescription method (optional)

The returned value will be used as the first sentence of the help information of the subcommand to describe the subcommand. If this method is not overridden, the returned value of getDescription will be used.

### Override the getFullDescription method (optional)

The returned value will be used in the help information of the subcommand to fully explain the subcommand. This can be a long passage or even several passages. If this method is not overridden, an empty string will be used, and the explanation will be empty.

### Override the prepareOptions method

### Override the parse method

Command subclasses can do command-line argument parsing and validation, which the Apache Commons CLI library isn’t able to do, in this method. For example, some option values must be converted from String to other types such as int and float; some number values must lie in a specific range (such as MAF should be 0~0.5); some options are incompatible and cannot be set together; and so on. If any error of the command-line arguments is detected, a CommandArgumentException must be thrown, which the framework will catch and process; otherwise a CommandArguments subclass instance must be created as the returned value to pack the arguments.

Note that don’t print any log with gear.util.Logger in this method, because the Command base class won’t set the log filename to the value of --out until this method is ended.

### Override the createCommandImpl method

This method must create and return the corresponding CommandImpl partner. The framework will then call execute on the CommandImpl instance to finally execute the subcommand.

## Register the New Subcommand into Gear

This is simple. In the constructor of gear.Gear, just create an instance of the new Command subclass and call addCommand with the instance to register the new subcommand.