# Overview

The general architecture of the quash mirrors many other parsing systems. A rudimentary parser will attempt to parse the inbound string (be it from terminal or a file) and attempt to create a data structure containing a list of commands to execute and everything that is needed to do so for each command. The arguments and environment become self-contained, making execution trivial. The parse makes extensive use of the built-on tokenizer to build structures as necessary. The tokenizer parses on pipes, spaces, new lines and redirects as necessary. Everything is copied to temporary buffers to avoid overwriting data. Static sizes are used wherever possible to minimize the chances of memory leaks. Each possible combination of commands (pipes, redirects, etc.) has its own function dedicated to handling it. For example, the basic command with arguments has a command that handles nothing else. The system call execvpe is used because it handles searching the PATH for the executable automatically, handling absolute paths and inheriting the environment in one call. Background calling is handled by determining whether or not to wait for the process in question to finish executing before continuing running the shell.

# Built-In Commands

Built in commands are handled simply by intercepting the parser before it can handle any other characters. For example, if the line buffer has “exit” the shell will return 0 and exit on the spot. For changing directories, a function change\_dir is called and is given the current line buffer. It then determines what arguments (if any) are present, builds a final path and then issues a chdir system call.

# Jobs

Job handling is implemented in a very rudimentary fashion. When a command is issued to the background, the function handling that command will write to a global array containing job information. When a process is finished (as signaled by the waitpid system call).

<Add More Stuff Here>