一、builtins.c中的函数实现

1. **builtin_cmd:**

   主要功能是读入一个cmd，并判断是否为内建命令。如果是，则跳转至对应命令函数，并返回1；如果不是，则返回0。在eval函数中，会将读入的命令发送到该函数处理。

   用简单的判断语句，实现命令的执行功能👇

```
if (strcmp(cmd->argv[0], "quit") == 0)
    ...
else if (strcmp(cmd->argv[0], "cd") == 0)
    ...
...
else
    return 0;
return 1;
```

2. **do_bgfg:**

   主要功能是具体实现bg命令和fg命令。

   首先判断输入的合法性，如果输入指令数不等于2，则会报错并返回👇

```
    if (cmd->argc != 2) {
        fprintf(stderr, "%s command requires PID or %%jobid argument\n", cmd-
>argv[0]);
        return;
    }
```

   在输入合法的前提下，判断指令类型（jobid/pid），然后依次判断输入是否为整数、指定job是否存在，最终将指定job记录在变量[job]中。以job id为例👇

```
    if (cmd->argv[1][0] == '%') {  // 使用 job id 格式
        int jid;
        if (sscanf(cmd->argv[1], "%%%d", &jid) != 1) {  //判断是否为整数
            fprintf(stderr, "%s: argument must be a PID or %%jobid\n", cmd-
>argv[0]);
            return;
        }
        job = get_job_by_jid(jobs, jid);  //判断 job 是否存在
        if (!job) {
            fprintf(stderr, "%%%d: No such job\n", jid);
            return;
        }
    }
```

   找到指定[job]后，分别按照指令类型（bg/fg）进行处理，包含执行和报错功能。以bg为例，先将暂停的job恢复运行，再设置状态为后台，如果出问题则报错👇

```
    if (strcmp(cmd->argv[0], "bg") == 0) {
        if (kill(-job->pid, SIGCONT) < 0)  //恢复运行
            fprintf(stderr, "bg: %s\n", strerror(errno));
        else {
            job->state = BG;  //设置状态为后台
            printf("[%d] (%d) %s &\n", job->jid, job->pid, job->cmdline);
        }
    }
```

3. **do_cd:**

   主要功能是具体实现cd命令。

   先判断输入的合法性，在输入合法的基础上切换工作目录至指定位置。以下是目录切换的功能👇

```
        char cwd[MAXLINE];  //初始化，存储工作目录
        if (getcwd(cwd, sizeof(cwd)) != NULL)  //获取当前路径
            setenv("PWD", cwd, 1);  //更换环境变量
```

4. **do_kill:**

   主要功能是关掉指定job。

   先判断输入合法性（同上，不再赘述），确认该进程合法且存在的情况下，将它关掉，并输出操作成功的报告👇

```
    if (kill(-job->pid, SIGKILL) < 0) {
        fprintf(stderr, "(%d): No such process\n", job->pid);
    }
    else {
        printf("Job [%d] (%d) terminated by signal %d\n", job->jid, job->pid,
SIGKILL);
        delete_job(jobs, job->pid);
    }
```

二、signals.c中的函数实现

1. **sigchld_handler:**

   主要功能是暂停某个进程的具体操作

   需要使用循环将所有待回收的子进程进行回收👇

```
    while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) {  //循环回
收所有
        job_t *job = get_job_by_pid(jobs, pid);
        if (!job)
            continue;

        if (WIFEXITED(status))  //正常退出：删除作业
            delete_job(jobs, pid);
```

```
        else if (WIFSIGNALED(status)) {  //因信号终止：输出消息，再删除作业
            ...
        }
        else if (WIFSTOPPED(status)) {  //被停止：输出消息，并更新状态
            ...
        }
    }
    errno = olderrno;  //恢复原始状态
```

2. **sigint_handler和sigtstp_handler**

   主要功能是终止和暂停前台作业👇

```
void sigint_handler(int sig) {  //终止前台作业
    pid_t pid = fg_pid(jobs);
    if (pid > 0)
        kill(-pid, SIGINT); //发送到进程组
}

void sigtstp_handler(int sig) {  //暂停前台作业
    pid_t pid = fg_pid(jobs);
    if (pid > 0)
        kill(-pid, SIGTSTP);
}
```

三、shell.c

主要功能是解析和执行用户输入的命令

首先读入命令，并且判断是否为空👇

```
    if (parse_command_line(cmdline, &cmd, &bg) != 0)  //解析命令行，构造 command_t
链表
        return;
    if (cmd == NULL || cmd->argv[0] == NULL) {  //若命令为空则直接返回
        if (cmd)
            free_command(cmd);
        return;
    }
```

对于内建命令，则直接调用上述函数进行执行👇

```
    if (builtin_cmd(cmd)) {  //内建命令直接在父进程中处理
        free_command(cmd);
        return;
    }
```

然后调用fork函数，根据返回进程是子进程还是父进程进行后续操作，以子进程部分为例👇

```
    if (pid == 0) {  //子进程，执行外部命令
```

```
        setpgid(0, 0);  //将子进程放到新进程组，便于信号管理
        signal(SIGINT, SIG_DFL);
        signal(SIGTSTP, SIG_DFL);
        signal(SIGCHLD, SIG_DFL);  //恢复默认信号处理

        if (cmd->infile) {  //处理输入
            int fd = open(cmd->infile, O_RDONLY);  //打开输入文件
            if (fd < 0) {  //失败了
                ...
            }
            dup2(fd, STDIN_FILENO);  //把输入复制到标准输入
            close(fd);
        }

        if (cmd->outfile) {  //处理输出
            ...
        }

        if (execvp(cmd->argv[0], cmd->argv) < 0) {  //调用execvp执行外部命令
            fprintf(stderr, "%s: Command not found\n", cmd->argv[0]);
            exit(1);
        }
    }
```

再将父进程的功能写入，根据作业属于前台还是后台分别处理👇

```
        else if (pid > 0) {  //父进程，处理前台/后台作业

        add_job(jobs, pid, bg ? BG : FG, cmdline);

        if (!bg) {  // 前台作业
            tcsetpgrp(STDIN_FILENO, pid);  //将终端控制权交给子进程组
            waitfg(pid);  //等待前台作业完成
            tcsetpgrp(STDIN_FILENO, getpgrp());  //将终端控制权还给shell
        }
        else {  // 后台作业
            job_t *job = get_job_by_pid(jobs, pid);
            printf("[%d] (%d) %s &\n", job->jid, job->pid, job->cmdline);
            fflush(stdout);
        }
    }
```

最后释放内存，函数结束