<현재 디렉토리 명을 쉘에 표시>

```
p2023350225@p2023350225:~/shell$ make run
./shell
/home/p2023350225/shell$
```

현재 디렉토리 위치를 표시하고 있다.

```
char cwd[MAX_CMD_LEN];
if (getcwd(cwd, sizeof(cwd)) != NULL) {
    printf("%s$ ", cwd);
}
```

Getcwd 함수가 현재 작업 디렉토리를 받아온다.

<일반적인 리눅스 bash 쉘의 사용자 인터페이스 형태를 띌 것>

바로 위의 사진에서 보이듯이, 일반적으로 생각할 수 있는 (디렉토리 위치)$ 뒤에 커맨드를 넣을 수 있는 인터페이스로 만들었다.

<cd 명령어>

```
p2023350225@p2023350225:~/shell$ make run
./shell
/home/p2023350225/shell$ cd /home
/home$ cd ./p2023350225
/home/p2023350225$
```

폴더를 이동하는 cd 명령어다. Cd 커맨드의 문법이 정상적으로 작동하는 것을 확인할 수 있다.

```
void change_directory(char *path) {
    if (chdir(path) != 0) {
        perror("cd failed");
    }
}
```

Chdir 명령어를 이용해 경로를 바꾸게 하였고, 존재하지 않는 경로면 실패 메시지를 띄운다.

<pwd 명령어>

```
/home/p2023350225/shell$ pwd
/home/p2023350225/shell
/home/p2023350225/shell$ cd /home
/home$ pwd
/home
```

Pwd는 현재 작업 경로를 출력하는 명령어다. Pwd 명령어 다음 줄에 현재 경로가 그대로 잘 찍히는 것을 확인할 수 있다.

```
void print_working_directory() {
    char cwd[MAX_CMD_LEN];
    if (getcwd(cwd, sizeof(cwd)) != NULL) {
        printf("%s\n", cwd);
    }
        else {
        perror("pwd failed");
    }
}
```

앞에서도 사용한 getcwd 함수를 이용해서, 그로 받아온 cwd 변수에 들어있는 경로를 출력하게 하였다.

<파이프 라인>

```
pipe(pipe_fd);
pid1 = fork();
if (pid1 == 0) {
    close(pipe_fd[0]);
    dup2(pipe_fd[1], STDOUT_FILENO);
    close(pipe_fd[1]);

    char *args[MAX_ARGS];
    parse_command(command, args);
    execvp(args[0], args);
    exit(1);
}

pid2 = fork();
if (pid2 == 0) {
    close(pipe_fd[1]);
    dup2(pipe_fd[0], STDIN_FILENO);
    close(pipe_fd[0]);

    char *args[MAX_ARGS];
    parse_command(command, args);
    execvp(args[0], args);
    exit(1);
}
```

명령어를 파이프라인 기호와 함께 처리하려고 했으나 제대로 완성하지 못하였다.

<다중 명령어>

```
void handle_multiple_commands(char *command) {
    char *token = strtok(command, ";");
    while (token != NULL) {
        execute_command(token);
        token = strtok(NULL, ";");
    }
}
```

; 기호로 명령어를 나누어 실행하려고 했는데, 제대로 완성하지는 못했다.

<백그라운드 실행>

```
void run_in_background(char *command) {
    pid_t pid = fork();
    if (pid == 0) {
        char *args[MAX_ARGS];
        parse_command(command, args);
        execvp(args[0], args);
        exit(1);
    }
}
```

자식 프로세스의 실행 동안 부모 프로세스는 종료를 기다리지 않도록 했다. 그러나 제대로 개선하지 못해 완성에 실패했다.

<exit 입력>

```
p2023350225@p2023350225:~/shell$ make run
./shell
/home/p2023350225/shell$ cd /home
/home$ cd ./p2023350225
/home/p2023350225$ cd ./shell
/home/p2023350225/shell$ exit
Exiting shell...
p2023350225@p2023350225:~/shell$
```

Exit 명령어를 입력하면 다음과 같은 문구가 나오고 다시 원래 인터페이스로 돌아온다.

```c
void handle_exit() {
    printf("Exiting shell...\n");
    exit(0);
}
```

Exit(0)으로 프로그램을 종료하게 하였다.

<추가 명령어 – ls>

```
p2023350225@p2023350225:~/shell$ make run
./shell
/home/p2023350225/shell$ ls
Makefile  README.md  shell  shell.cpp  shell.o
/home/p2023350225/shell$ ls -l
total 44
-rw-rw-r-- 1 p2023350225 p2023350225   149  5월 12 15:46 Makefile
-rw-rw-r-- 1 p2023350225 p2023350225     7  5월 12 15:32 README.md
-rwxrwxr-x 1 p2023350225 p2023350225 14856  5월 12 19:06 shell
-rw-rw-r-- 1 p2023350225 p2023350225  5361  5월 12 19:05 shell.cpp
-rw-rw-r-- 1 p2023350225 p2023350225  9352  5월 12 19:06 shell.o
/home/p2023350225/shell$
```

추가로 구현한 ls 명령어다. Ls 명령어는 현재 디렉토리의 파일 목록을 확인하며 -l 옵션은 long list format으로 출력하게 한다.

```c
void handle_ls(char *command) {
    char *args[MAX_ARGS];
    parse_command(command, args);
    pid_t pid = fork();
    if (pid == 0) {
        execvp("ls", args);
        exit(1);
    } else {
        wait_for_child();
    }
}
```

인자를 받은 후 execvp로 ls 명령어를 실행하게 하였다.

<추가 명령어 – echo>

```
/home/p2023350225/shell$ echo Hello, World!
Hello, World!
/home/p2023350225/shell$ echo The current directory is $(pwd)
The current directory is /home/p2023350225/shell
```

Echo 명령어는 주어진 메시지를 출력한다. 사진의 두 번째 명령에서는 $(pwd)를 함께 넣어 현재 위치 경로를 문장과 함께 출력하게 하였다.

```
void handle_echo(char *command) {
    char *args[MAX_ARGS];
    parse_command(command, args);
    for (int i = 1; args[i] != NULL; i++) {
        printf("%s ", args[i]);
    }
    printf("\n");
}
```

명령어와 인자를 적절히 파싱하여 출력하게 하였다.

<보안위협에 대한 고려사항>

```
void wait_for_child() {
    pid_t pid = wait(NULL);
    if (pid == -1) {
        perror("Wait failed");
    }
}
```

자식 프로세스 종료를 기다리며 좀비 프로세스를 방지하도록 하였다.