Unix Shell

사용자가 운영체제와 상호 작용하기 위한 명령 인터페이스. 대화식(interactive)명령 프롬프트 / 프로그래밍 환경으로 사용된다.

interactive shell

명령 입력시 바로 출력하는 명령어

dual nature

- An interactive command prompt
- A programming environment -> 루비, 함수 등 처리 가능
- interactive 명령 프롬프트
- 프로그래밍 환경

Interactive facility

- command aliasing (ex. cd, ls) = Built-in Commands
  cd: 디렉토리 변경하는 명령어의 aliase
- 최근 명령의 호출 및 수정과 같은 다양한 interactive한 기능 제공한다.

Shell as Programming Environment

- 변수
- 루프
- 함수
- 예외
  --> 프로그램을 짤 순 있지만 권장되지 않음

스크립트와 프로그램을 생성하는 프로그래밍 환경으로 사용된다.

Words

Shell은 모든 input을 워드 (String)으로 취급한다. -> ', ", \ 등으로 단어 파싱 처리

- single statement 앞 문장 수행되어야 다음이 실행되며 마지막은 엔터(&&)로 끝난다.

문장 파싱 후, shell이 결정하는 것

- 변수 할당
- builtin command
- if, while같은 control statement가 있는지
- 외부 프로그램이 있는지..

Builtin Commands

== command alliasing
외부 프로그램을 실행하지 않고 셸 내에서 실행되는 명령이다.
셸의 내부 상태를 변경하며 cd, ls같은 명령어가 존재한다.

Builtin 명령어 사용시 외부 프로그램 실행하지 않으며 intercode에서만 동작함

- 효율성(간단하게 실행되니까)
- shell internal state must be changed
  현재 디렉토리 경로/환경변수 등등 프로그램 실행 전 변경하고 접근한다.

internal state를 변경하는 건 fork후에 하지 못하기 때문에 built-in command로 해결한다.

External Commands

셸에서 builtint command가 아니면 전부 외부 커맨드로 간주한다.
셸은 새로운 프로세스를 포크하고 exec를 사용해 외부 명령을 실행한다.

첫 단어는 실행할 바이너리이고 다음부터는 함수의 아규먼트이다.

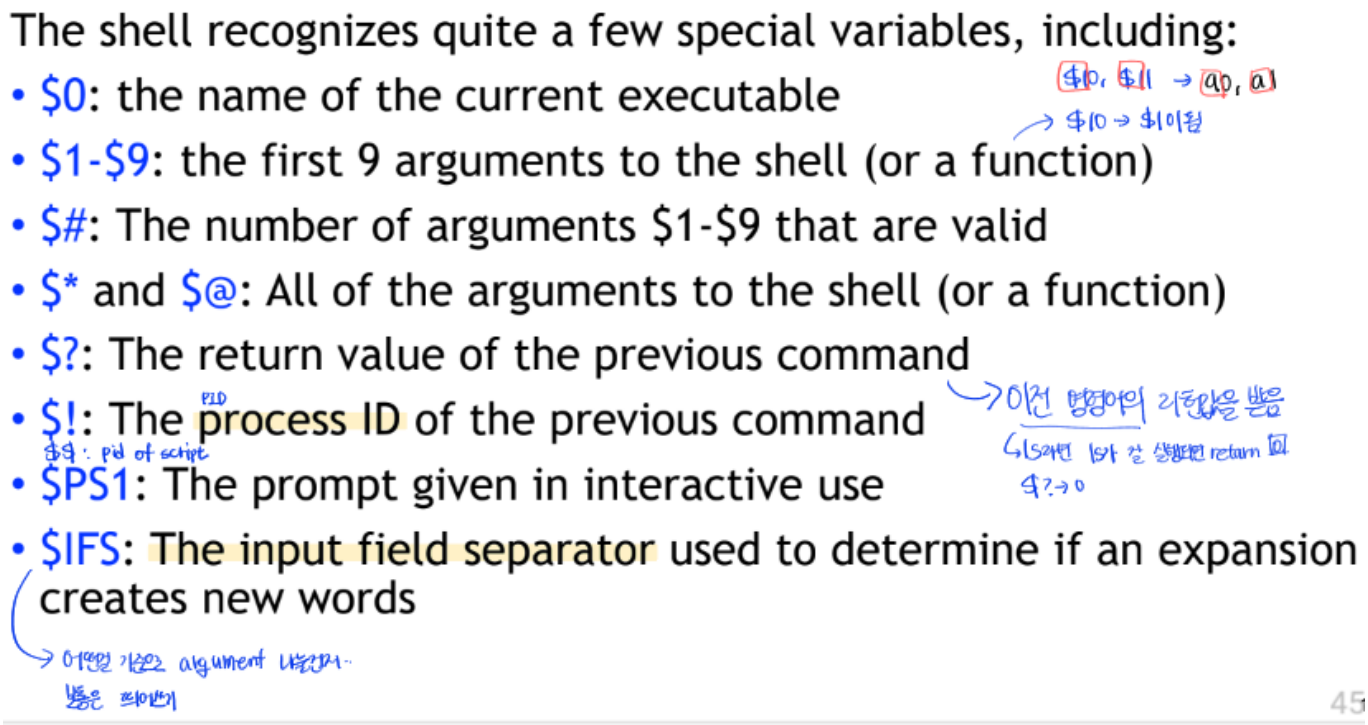
```
./hello 127.0.0.1 9090
```

변수

- 변수는 모두 문자열이다.
- 앞 뒤로 띄어쓰기 X

  ```
  VAR=value
  ```

스페셜 변수



- IFS Input Feld Separator
  어떤 것을 기준으로 argument를 나눌 것인지 설정
  보통은 띄어쓰기로 구분함

```
$ VAR="arg1 arg2"
$ ./writeargs $VAR
./writeargs
arg1
arg2
# 띄어쓰기 때문에 분리되어 나옴
```

- Variable Interpolation 변수를 실제 함수의 변수에 할당함

```
echo $Name
```

- Command Interpolation 커맨드의 결과를 커맨드하는 것..

```
current_dir=$(pwd)
# pwd: built-in command
```

pwd: 현재 디렉토리를 출력함
-> current_dir 변수에 저장됨

- Shell vs Environment 변수

```
# PATH 변수에 현재 디렉토리 추가
export PATH=$(pwd):$PATH
```

현재 디렉토리를 PATH 환경변수에 추가하여 현재 디렉토리에 있는 실행 파일을 전체 경로로 지정하지 않고도 실행할 수 있음.

./hello
hello
동일하게 작동함 (절대경로를 지정하지 않아도 되기때문에)

- Environment
  환경변수 설정: export var [VAR2...]
  setenv() or putenv()

- Globbing 정규표현식으로 매치하는 것 찾기..

  - *: 113* -> 113으로 시작하는 문자열
  - ?: 적어도 하나 매치
  - []: 집합에 포함된 것

  *.c -> .c로 끝나는 모든 파일
  *.? -> file.txt (o) / file. (x) / .c (x) *.[ch] -> *.c or *.h

- Pipe and Redirection

  - pipe: '|'
  - dup2로 fd 열기: <, <<, >, >>
    꺽쇠 하나는 overwrite
  - dup2로 copy: >&

- Simple File Redirection < file: stdin > file: 출력 넘기기 2(fd: stderr)>file: 에러 로그

```
echo "Hi">file.txt
# 출력 안되고 file.txt에 저장됨
```

- Use Redirection

```
# word 개수 세기
wc -w < file

# 단위로 분리
cut -d' ' -f5 > totals.txt
```

공백을 구분자로 사용하여 필드 분리 후 다섯번째 필드를 totla에 저장함

- Use Pipe

```
cmd1|cmd2
# 두개 사이의 pipe 생성됨
```

1. pipe 생성
2. cmd1 포크, cmd2 포키
3. dup2 사용 pipefd[1]: cmd1 -> stdout pipefd[0]: cmd2 -> stdin 자동 지정됨
4. exec 호출
5. cmd2 끝나길 기다림

```
1. Create pipe()
2. fork twice
3. use dup2() to connect
    - pipefd[1] to stdout(1) of cmd1
    - pipefd[0] to stdin(0) of cmd2
4. call exec() in child -> exec(프로그램 실행)
5. wait for cmd2 to exit
```

- Duplication Descriptor 셸은 새로운 파일을 열기 않고도 descriptor를 중복시킬 수 있다.

```
N>&M #dup2(N,M)
echo Could not open file 1>&2 # 해당 string을 stderr로 출력할 수 있다.
```

Summary

- Shell은 거의 직접적으로 시스템콜을 호출할 수 있다.
    - fork/exec
    - open/close
    - read/write
    - dup2
    - wait
- Shell은 명령 프롬프트와 프로그램 환경을 제공한다.

1. An interactive command prompt
2. A programming environment