1. Introduction to the Command Line

1.1 Definitions

1. 유닉스 계열 시스템

유닉스 계열(Unix-like)은 유닉스와 비슷하면서 유닉스가 아니다라는 뜻으로 유닉스와는 별개의 용어이다. 유닉스가 아님에도 기능적으로 유닉스 규격에 호환되어 유닉스의 대체품으로 쓸 수 있는 유닉스와 비슷한 운영 체제를 말한다. 대표적으로 리눅스(Linux)가 있다.

2. 그래픽 사용자 인터페이스(GUI)

그래픽 사용자 인터페이스는 사용자가 편리하게 사용할 수 있도록 입출력 등의 기능을 알기 쉬운 아이콘 따위의 그래픽으로 나타낸 것이다.

3. 명령 줄 인터페이스(CLI)

커맨드 라인 인터페이스 또는 명령 인터페이스는 가상 터미널 또는 터미널을 통해 사용자와 컴퓨터가 상호작용하는 방식을 뜻한다.

4. 유틸리티 소프트웨어(Utility Software)

컴퓨터 소프트웨어의 하나로, 컴퓨터의 동작에 필수적이지는 않지만 컴퓨터를 이용하는 주 목적에 대한 부차적인 일부 특정 작업을 수행하는 소프트웨어를 두루 가리킨다. 유틸리티 소프트웨어는 컴퓨터를 분석, 구성 최적화 또는 유지 관리하도록 설계된 시스템 소프트웨어이다.

5. 단말 에뮬레이터(Terminal Emulator)

단말 에뮬레이터 또는 터미널 에뮬레이터는 몇 가지 다른 디스플레이 구조를 갖춘 덤브 비디오 단말기를 가상으로 구현하는 프로그램을 말한다. 그래픽 사용자 인터페이스상에서 어플리케이션으로 작동하는 단말 에뮬레이터를 터미널 윈도우라고 한다.

6. 셸(Shell)

셸은 운영체제 상에서 다양한 운영체제 기능과 서비스를 구현하는 인터페이스를 제공하는 프로그램이다. 셸은 사용자와 운영체제 내부 사이의 인터페이스를 감싸는 층이기 때문에 그러한 이름이 붙었다. 셸은 일반적으로 명령 줄과 그래픽 형의 두 종류로 구분된다. 명령 줄 셸은 운영체제 상에서 명령 줄 인터페이스(CLI)를 제공한다.

7. 명령 프롬프트(Command Prompt)

명령 프롬프트는 컴퓨터 터미널 또는 터미널 에뮬레이터의 명령 줄 대기모드를 가리킨다. 일반적으로 커맨드 프롬프트 또는 셸 프롬프트 라인은 아래의 형식을 가진다.

```
user@host: /home$
user@host: /home$
```

8. 명령 언어(Command Line)

명령 언어는 컴퓨터 작업 제어를 위한 도메인 고유의 해석 언어이다. 명령줄 인터페이스에서 작성되며, 명령 언어의 일반적인 예는 셸 또는 배치 프로그래밍 언어이다.

9. 파일 시스템(Filesystem)

파일 시스템은 우리의 파일을 조작하는 시스템으로 디렉토리라는 그룹에 정돈되게 보관하는 시스템이다.

1.2 Command Prompt

셸에서 명령을 입력할 때 가장 기본적인 형태는 다음과 같다.

```
user@host: /home$ command -options arguments
```

명령어는 명령어 이름, 옵션, 매개변수로 구성되어 있다. 옵션은 명령어의 행위를 지정할 수 있으며 하나의 '-'를 사용하는 short option과 '--'를 사용하는 long option이 존재한다. long option의 경우 사람이 읽기 쉽게 표현되어 있다. 하나 이상의 명령어를 붙여 복수의 옵션을 지정할 수도 있다.

```
# Show previous commands
user@host: /home$ history

# Clear Command Prompt
user@host: /home$ clear

# Exit Command Prompt
user@host: /home$ clear
```

2. The Filesystem

2.1 Filesystem

파일 시스템(Filesystem)은 우리의 파일을 조작하는 시스템으로 Windows의 Explore, OS X의 FInder 등이 있다. 파일 시스템을 통해 파일을 **디렉토리(directory)**라고 불리는 저장소에 보관할 수 있다. 디렉토리는 서로 다른 디렉토리들로 구성될 수 있으며, 가장 상위에 있는 디렉토리를 **루트(root)**, 하위에 있는 디렉토리를 **하위 디렉토리(subdirectory)**라고 한다.

유닉스 계열 시스템은 디렉토리가 나무와 같이 뻗어나가며 상위 디렉토리와 하위 디렉토리로 구성되어 있다. 이와 같은 관계를 **위계 디렉토리 구조(hierachical directory structure)**이라고 한다. 그리고 명령 프롬프트에서 명령 줄에 입력되어 있는 경로가 **현재 작업 경로(Current Working Directory)**라고 한다.

유닉스 계열은 **모든 것은 파일(Everything is file)**이라는 가치 아래 작동하고 있다. 즉, 명령, 파일, 장치, 디렉토리 등 모든 것은 파일로 표현되며 여러 개의 하위 파일과 디렉토리 주소들을 담고 있다는 것이다.

2.2 Path

대부분의 파일과 프로그램이 생성되고 저장되는 공간을 **홈 디렉토리(Home directory)**라고 한다. 경로의 종류는 상대경로와 절대경로가 있다. **절대 경로**는 루트 디렉토리에서 원하는 디렉토리까지 모든 디렉토리를 기입하는 방식이고, **상대 경로**는 현재 작업 경로를 기준으로 원하는 디렉토리까지 기입하는 방식이다.

```
# Print Current Working Directory
user@host: /home$ pwd
user@host: /home$
```

2.3 Directory Commands

현재 디렉토리의 상태를 확인하고 다른 디렉토리로 이동하는 대표적인 명령어는 ls와 cd가 있다. ls 명령어는 원하는 경로의 디렉토리, 파일 등을 옵션에 따라 다양하게 보여준다. cd 명령어의 경우 디렉토리의 위치를 변경할 때 사용된다.

```
# Listing the non-hidden contents of the current directory
user@host: /home$ ls
# Listing the non-hidden contents of path
user@host: /home$ ls /home/...
# Listing the non-hidden contents of the current directory in long format
user@host: /home$ ls -l
# Listing all contents of the current directory
user@host: /home$ ls -a

# Change to directory
user@host: /home$ cd /home/dir1
# Change to home directory
user@host: /home$ cd
user@host: /home$ cd ~
# Change to Parent directory
user@host: /home$ cd ../../
```

3. Modifing the filesystem

3.1 Create

```
user@host: /home$ mkdir [dir_name]
user@host: /home$
```

3.2 Remove

rmdir은 empty directory만 삭제 가능하며, non-empty directory의 경우 rm -R 명령어를 통해 제거할 수 있다.

```
user@host: /home$ rmdir [dir_name]
user@host: /home$ rm -R [dir_name]
```

3.3 Copy

-i 옵션은 중복되는 파일명에 데이터를 복사하여 덧씌울지를 결정하는 옵션이다. -R 옵션은 디렉토리에 한정해서 하위 폴더들을 모두 재귀적으로 복사하는 옵션이다.

```
user@host: /home$ cp -i [path/dir_name] [path/dir_name]
user@host: /home$ cp -R [path/dir_name] [path/dir_name]
```

3.4 Move

```
# Moving
user@host: /home$ mv [path/dir_name] [path/dir_name]
# Renaming
user@host: /home$ mv [path/dir_name] [path/new_name]
```

4. Global Patterns and Wildcards

4.1 Global Patterns

정규 표현식(Regular Expression)은 특정한 규칙을 가진 집합을 표현하는 데 사용하는 형식 언어이다. 정규 표현식이라는 문구는 일치하는 텍스트가 준수해야 하는 "패턴"을 표현하기 위해 특정한 표준의 텍스트 신택스를 의미하기 위해 사용된다. 글로벌 패턴은 POSIX 표현식과 정규 표현식을 사용하여 복수의 문자열의 집합을 정의하는 과정이다.

4.2 Escaping character

정규 표현식이나 POSIX 표현식으로 문자열이 표현되지 않고 문자 그대로 파일명이 abc?.txt인 경우 특수문자 앞에 backslash()를 붙여줌으로써 특수문자가 그대로 읽혀질 수 있도록 한다.

4.3 Wildcards

와일드카드(Wildcard)는 컴퓨터에서 특정 명령어로 명령을 내릴 때, 여러 파일을 한꺼번에 저장할 목적으로 사용하는 기호를 가리킨다. 주로 특정한 패턴이 있는 문자열 혹은 파일을 찾거나 긴 파일명을 생략할 때 사용된다.

Pattern	Meaning
?	단일 글자를 매칭함
*	하나 이상의 문자열을 매칭함
[list_of_characters]	[list_of_characters]안에 있는 어떤 문자라도 매칭함
[!list_of_characters]	[!list_of_characters]안에 있는 어떤 문자를 제외하고 매칭함
[[:alpha:]]	알파벳을 매칭함
[[:digit:]]	숫자를 메칭함
[[:alnum:]]	알파벳, 숫자를 매칭함
[[:lower:]]	알파벳 소문자를 매칭함
[[:upper:]]	알파벳 대문자를 매칭함

1. wildcards 내부에 있는 파일을 정리하기 위한 htm_files archive data 디렉토리를 생성하라
2. html 파일을 html_files directory로 이동시켜라
3. 19년도 데이터가 아닌 csv파일을 archive로 이동시켜라
4. 나머지 파일은 data로 이동시켜라

```
user@host: /home$ ls -a
user@host: /home$ mkdir html_files archive data
user@host: /home$ mv *.html html_files
user@host: /home$ mv 201[!9]?.csv archive
user@host: /home$ mv *.csv data
```

4.4 Find

find 명령어는 전체 디렉토리에서 정규표현식을 만족하는 파일명을 검색한다.

```
user@host: /home$ find [location] -name [filename]
```

5. Users and Permmisions

5.1 Owner

사용자(User)는 컴퓨팅 환경에서 컴퓨터 시스템을 사용하는 사람을 일컫는 말이다. 사용자들은 보안, 로그인 기록, 리소스 관리 등의 목적을 위해 인증을 시도해야 한다. 인증을 시도할 때 사용자는 계정과 사용자 이름을 가지고 있어야 하며 이를 이용해 사용자 인터페이스를 사용하여 시스템에 접근하고 인증을 처리한다. 하나의 네트워크에는 수많은 사용자들이 연결디어 있어, 사용자들을 분리하고 컴퓨터의 무결성(integrity)를 보장하기 위해, 사용자라는 개념이 사용되게 되었다.

5.2 Check Owner

사용자를 확인하는 명령은 다음과 같다.

```
user@host: /home$ whoami
user@host: /home$ id
user@host: /home$ group
user@host: /home$ stat [filename]
```

유닉스 계열 시스템은 사용자들을 대표적으로 3그룹으로 나누어 구분한다.

- uid : User id의 약어로 사용자별 고유한 값을 가지고 있다
- gid : Group id의 약어로 uid가 속해있는 Group의 id이다.
- sudo : 유닉스 및 유닉스 계열 운영 체제에서, 다른 사용자들의 보안 권한 및 프로그램을 구동할 수 있도록 하는 슈퍼유저

5.3 Change Owner

chown 명령어는 파일과 디렉토리의 owner를 변경한다.

```
user@host: /home$ chown [new_owner][:new_group] [path/file]
```

5.4 Permissions

권한(Permissions)는 사용자가 할 수 있는 행동의 집합을 의미한다. 따라서 사용자가 다른 사용자의 디렉토리와 파일을 조작하고 변경하는 것을 방지한다.

권한은 Owner, Owner Group, Everyone에 대해 정의된다. 각각의 사용자들은 고유한 **파일 접근 모드(file access mode)**를 가지게 된다. 파일 접근 모드는 **rwx(read mode, write mode, execute mode)**로 정의되며 ls -l 명령어를 실행했을 때 '-r-xr--r--'과 같은 문자열로 표현된다. 각각의 사용자 그룹이 어떤 파일 접근 모드를 가지 있느냐에 따라 다른 사용자가 접근하는 것을 막을 수 있다.

앞서 이진법 체계에서 언급했던것과 같이 파일의 권한은 8진법의 체계에 따라 파일 접근 모드를 정의할 수 있다.

file access mode	octal	meaning
---	0	no permissions
--x	1	execute only permission
-w-	2	write only permissions
-wx	3	write and execute permissions
r--	4	read only permissions
r-x	5	read and execute permissions
rw-	6	read and write permissions
rwx	7	read, write, and execute permissions

5.5 Change Permissions

chmod 명령어는 파일과 디렉토리의 권한을 변경하는데 사용된다. 8진법 체계에 따라 숫자로 간단하게 파일 접근 모드를 정의할 수 있고, [ugoa][+-=][rwx] 옵션을 부여하여 각각의 그룹과 변경 사항 변경내용을 명시할 수도 있다.

```
user@host: /home$ chmod [ugoa][+-=][rwx] [path/file]
```

6. Getting Help and Reading Documentation

6.1 Type of Commands

유닉스 계열 시스템은 "모든 것은 파일"이라는 가치 아래서 작동한다. 명령어도 root 디렉토리의 하위 디렉토리에 파일로 존재한다. 대표적인 명령어의 종류는 다음과 같다

1. file : 주로 프로그램, 유틸리티, 도구 등으로 언급되며 간단하게 실행가능한 파일
2. bulitin : RAM 속에서 사용가능하며 빠르게 실행되어 시스템에서 troubleshooting 에 사용
3. alias : command의 축약어로 간결하게 축약된 단어의 형태로 사용
4. function : Shell language로 작성된 함수
5. keyword : 특정한 목적으로만 존재하며 다른 목적으로 존재할 수 없는 단어

```
user@host: /home$ type -[options] [filename or command]
```

리눅스 이전 유닉스 운영체제를 만들 때 용량 문제로 bin과 sbin 디렉토리를 여러 곳에 분산시켜 만들었는데, 현재 유닉스 기반 OS에서도 이러한 이유로 bin과 sbin 연관 디렉토리들이 파일 시스템에 남아있다.

1. bin : cd, ls 등의 사용자 커맨드 파일이 위치한 디렉토리
2. sbin : systemctl 등의 시스템 커맨드 파일이 위치한 디렉토리
3. usr/bin : 필요에 의해 설치된 사용자 커맨드 파일이 위치한 디렉토리
4. usr/sbin : 필요에 의해 설치된 시스템 커맨드 파일이 위치한 디렉토리
5. usr/local/bin : 기타 사용자 커맨드 파일이 위치한 디렉토리
6. usr/local/sbin : 기타 시스템 커맨드 파일이 위치한 디렉토리

6.2 Alias using of Commands

자주 사용하는 명령어는 alias 명령어를 통해 축약할 수 있다.

```
user@host: /home$ alias [alias_name] = [command+name]
user@host: /home$ unalias [alias_name]
```

6.3 Priority of Commands

명령어의 우선순위는 다음과 같다.

1. Alias
2. Special bulitin
3. Functions
4. Regular builtin
5. PATH + file

6.4 Reading Documents

명령어는 수많은 옵션들과 매개변수를 입력할 수 있다. 따라서 문서를 참조해야 할 필요가 있는 경우 함수의 종류에 따라 서로 다른 명령어를 사용할 수 있다. 명령어나 프로그램에 대한 문서를 셸에서 열람할 경우 man 혹은 help 명령어를 통해 가능하다. help 명령어는 bulitin 명령어에 대해서 간단한 문서를 제공한다.

man 명령어의 출력 결과 페이지를 넘길 수 있는 간단한 텍스트 문서가 제공된다. 이때 생성된 상호작용 가능한 프로그램을 less 라고 하고, 보여지는 문서를 man page라고 한다. man page의 구성요소는 다음과 같다.

- NAME : 명령어의 이름과 간단한 설명
- SYNOPSIS : 문법
- DESCRIPTION : 옵션
- OPTIONS : DESCRIPTION에 옵션에 대한 정보가 없는 경우 제공

문서상의 옵션의 표기법은 다음과 같다.

- **bold text** : 표기된 대로 작성한다
- italic text : 적절한 argument로 대체한다
- [-abc] : [ ] 내부에 존재하는 argument 모두 사용 가능함
- -a|-b : '|' 로 구분되어 있는 옵션은 서로 같이 사용 불가능
- argument ... : argument는 반복 가능함
- [expressions] ... : [ ] 내부에 있는 표현은 반복 가능함

6.5 less

유닉스 계열의 less 명령어는 전체 파일을 page by page로 출력가능한 CLI tool이다.

```
user@host: /home$ less [option] [filename] ...
user@host: /home$
```

7. File Inspectation

텍스트 파일(Text file)은 파일, 코드, 데이터 형식도 저장 가능하다.

7.1 head/tail

head, tail 명령어는 텍스트 파일의 열을 순서대로 출력한다. head -n 5는 처음부터 다섯번째까지를, -n -10은 처음부터 마지막 열번째 까지행을 출력한다. tail -n 1은 뒤에서부터 첫번째 행을, -n +5는 뒤에서부터 앞에서 다섯번째까지의 행을 출력한다.

```
user@host: /home$ head [-n[-][K]] [filename]
user@host: /home$ tail [-n[+][K]] [filename]
```

7.2 wc

wc 명령어는 전체 파일의 행의 개수, 전체 단어의 수, 저장 공간의 크기를 출력한다. 행의 개수는 newline(\n) 구분자에 의해 생성되는 행의 개수이고, 단어의 수는 white space에 의해 구분되는 문자의 수, 저장 공간의 크기는 ASCII 인코딩의 bytes 크기를 의미한다.

```
user@host: /home$ wc [filename]
user@host: /home$
```

7.3 column

column 명령어는 텍스트 파일의 내부 데이터를 테이블 형식으로 정돈하여 출력한다. -s 옵션은 테이블의 열 구분자를, -t 옵션은 테이블을 tab(\t)에 의해 분리하여 출력한다.

```
user@host: /home$ columns [-s[]][-t] [filename]
user@host: /home$
```

7.4 shuf

shuf 명령어는 텍스트 데이터의 행을 랜덤 비복원 추출로 섞어 출력한다. -n K 옵션은 K개의 무작위 행을 추출한다.

```
user@host: /home$ shuf -n K [filename]
user@host: /home$
```

7.5 file

file 명령어는 파일의 종류를 출력한다. type은 명령어의 종류를 확인하지만 file은 모든 디렉토리와 파일의 종류를 출력한다.

```
user@host: /home$ file [filename]
user@host: /home$
```

8. Text Processing

Python 내부에서도 텍스트 처리를 진행할 수 있지만 Shell language를 활용한 텍스트 전처리는 파일 시스템과 직접적으로 상호작용하기 때문에 속도가 훨씬 빠르다.

8.1 cat/tac

cat 명령어는 한 파일의 데이터를 다른 파일의 데이터 위에 누적해서 출력한다. tac 명령어는 cat 명령어와는 반대로, 밑에서부터 데이터를 누적해서 출력한다. cat 명령어는 추가적인 argument 없이 단일 파일에 대해서 사용될 때 전체 파일을 출력하는 작업을 진행한다. 이는 ccat 명령어의 본래 목적과는 다르기 때문에 **cat abuse** 혹은 **useless use of cat** 이라고 한다.

```
user@host: /home$ cat [option] [file1] [file2] ...
user@host: /home$ tac [option] [file1] [file2] ...
```

8.2 sort

sort 명령어는 텍스트 파일의 데이터를 사전식으로 정렬한다. -r 옵션은 역순으로 데이터를 정렬하고, -u 옵션은 중복 데이터를 제거하고 데이터를 정렬한다. 특정 컬럼을 기준으로 데이터를 정렬할 때는 -s 옵션을 사용해서 구분자를 지정한뒤 -kn,m 으로 컬럼 범위를 지정해 준다(n~m). sort 명령어는 기본적으로 문자 데이터에 대해서 사전식으로 정렬하기 때문에 -k 옵션뒤에 g 옵션을 붙여 숫자로 정렬한다.

```
# Check dataset
user@host: /home$ column -s":" -t characters_no_header
user@host: /home$ sort -t":" -k3,3  -k4,4, -k5-8gr characters_no_header
```

8.3 cut

cut 명령어는 옵션으로 명시한 컬럼만을 추출한다. -d 옵션은 열 구분자를 명시하고 -fn,m은 n번째 열부터 m번째 열의 범위를 표시한다.

```
user@host: /home$ cut -d"," -f2,4,5,6 'Computers & Mathematics'
```

8.4 grep

grep 명령어는 정규 표현식을 만족하는 행을 추출한다.

- -n : 해당 패턴이 어떤 파일에서 매칭되는지 표기
- -v : 옵션은 패턴이 매칭되지 않는 행을 출력
- -i : 옵션은 정규표현식의 소문자 대문자 표현을 무시하고 인식
- -E : 옵션은 확장된 정규표현식을 사용하도록 한다.
- -h : 출력된 결과 앞에 있는 파일명을 제거함

```
user@host: /home$ grep -v '*9' characters_no_header
user@host: /home$ grep -i ',MATH' *
```

9. Redirection and Pipeline

9.1 Redirection

리다이렉션(Redirection)은 컴퓨팅 시스템에서 표준 스트림의 stdout을 사용자 지정 위치로 변경가능한 일반적인 명령어이다. > 연산자를 통해 위치를 지정할 수 있으며 명령어를 통해 수행한 결과를 파일로 저장하는 작업이다.

```
user@host: /home$ [command] > [filename]
```

9.2 Overwrite output into file

단순 리다이렉션 연산자는 기존 파일의 데이터를 지우고 내용을 다시 쓰게된다. 기존 데이터에 새로운 데이터를 추가하기 위해선 >> 연산자를 통해 데이터를 추가한다.

```
user@host: /home$ [command] > [filename]
user@host: /home$ [command] >> [filename]
```

9.3 Create empty file

파일을 생성하는 방법은 리다이렉션을 이용하는 방법과 nano 명령어, touch 명령어를 통해 텍스트 파일을 생성하는 방법이 있다. touch 명령어는 기존에 존재하는 파일의 생성시간을 변경하여 새로운 빈 파일을 만드는 방식이다.

```
# Making file with redirection
user@host: /home$ echo "blablabla" | [command] > [filename]

# Making file with nano
user@host: /home$ nano [filename]

# Making file with touch
user@host: /home$ echo -n "" > [filename]
user@host: /home$ touch [filename]
```

9.4 Pipeline

파이프라인(Pipeline)은 서로다른 프로세스의 표준출력을 표준 입력으로 전달하여 복잡한 명령어들의 연산을 가능하게 해준다. 파이프라인을 생성해서 명령어를 연결하는 것은 유닉스 계열 시스템에서 존재하는 프로그램만을 사용해서 복잡한 작업을 설계 가능하다.

```
# Check number of directores in /bin file $
user@host: /home$ ls -l /bin | tail -n+2 | wc -l
```

9.5 Null device

널 디바이스(Null Device)는 일부 운영체제에서 기록대상이 되는 모든 데이터를 버리지만 쓰기 작업은 성공했다고 보고하는 장치 파일이다. /dev/null 파일에 존재하며 어떠한 작업이 출력되는 내용을 보고 싶지 않을 때 이곳으로 출력을 stdout하면 아무것도 보여지지 않는다.

```
user@host: /home$ [command] 1> /dev/null
user@host: /home$ [command] 2> /dev/null
```

10. Standard Streams and File Descriptors

10.1 Process

스트림(Stream)은 데이터의 흐름을 말하며 진행중인 모든 프로세스들은 스트림을 통해서 환경과 상호작용하게 된다. **프로세스(Process)**는 실행중인 프로그램을 의미한다. 부모 프로세스는 다른 프로세스에 대해 선행되어 발생하는 프로세스이고, 자식 프로세스는 부모 프로세스 뒤에 실행되는 프로세스 이다.

10.2 Standard Streams

표준 스트림(Standard Streams)는 유닉스 계열 운영체제에서 컴퓨터 프로그램과 환경 사이에 미리 연결된 입출력 통로를 의미한다. 일반적으로 유닉스 계열 운영체제에서 동작하는 표준 스트림은 3가지가 존재한다.

- 입력 스트림(Standard input, stdin, 0)
- 출력 스트림(Standard output, stdout, 1)
- 오류 출력 스트림(Standard error, stderr, 2)

10.3 Redirecting/Piping Streams

프로세스를 리다이렉션하는 것은 stout을 file로 지정하는 것이다. 한 프로세스를 다른세스로 연결하는 파이프라인은 이전 프로세스의 stdout을 다음 프로세스의 stdin으로 연결하는 것이다. stderr의 리다이렉션은 stderr의 파일 디스크립터인 2를 > 앞에 붙여 2>로 리다이렉션 가능하다. 명령어는 입력을 stdin에서 가져온다.

```
user@host: /home$ [command] [option] [filename] 1> [filename]
user@host: /home$ [command] [option] [filename] 2> [filename]
user@host: /home$ [command] [option] [filename] 0< [filenmae]

user@host: /home$ [command] [option] [filename] | [command] [option] 1> [filename] 2> [filename]
```

10.4 File Descriptors

운영체제에서 표준 스트림을 인식하는 방식은 0, 1, 2의 정수로 표현하여 추상화 한다. 해당 정수를 **파일 디스크립터(File Descriptor)**라고 한다. 파일 디스크립터는 유닉스 계열 시스템에서 프로세스가 파일을 다룰때 이용하는 개념으로 프로세스에서 특정 파일에 접근할 때 사용하는 추상적인 값이 된다.

현재 작동하는 셸에 연결되어 있는 표준 스트림은 /dev/pts/2로, 어떤 프로세스가 작동하기 위해선 stdin을 셸에 연결하고, stdout과 stderr도 자동으로 셸에 연결되어 출력된다. 파일 스크립터를 사용하면 stderr의 목적지를 stdout과 동일하게 사용가능하다. >& 연산자는 한 표준스트림을 다른 표준스트림과 동일하게 지정한다.

```
# Save stdout and stderr in file1
user@host: /home$ [command] >file1 2>&1

# Print stderr on shell and save stdout in file1
user@host: /home$ [command] 2>&1 >file1
```

11. Working with Programs

11.1 Shell Variables

Python과 다르게 명령 언어를 통해 변수를 정의하는 방법은 대문자만을 사용해서 변수명을 설정한다. 셸은 빈칸을 기준으로 매개변수, 옵션을 구분하기 때문에 변수의 값에 빈칸이 존재하는 경우 ""로 해당 내용을 덧씌운다. 또한 변수명에 할당 연산자를 빈칸 없이 정의해야하는데, 이는 앞서 말한 이유와 동일하다

변수의 값을 불러오고 싶은 경우 echo 명령어와 접근 연산자인 $를 사용해 해당 변수의 값을 출력한다.

```
user@host: /home$ VARIABLE_NAME = "contents with blank"
user@host: /home$ echo $VARIASBLE_NAME
```

11.2 Environment Variable

환경 변수(Environment Variable)은 셸에서 실행하는 모든 프로그램에서 호출할 수 있는 변수이다. export 명령어를 통해 정의하며 Python Script 내부에선 os 패키지의 environ[] 메소드를 통해 호출할 수 있다.

```
user@host: /home$ export VARIABLE_NAME = "contents with blank"

user@host: /home$ python
user@host: /home$ import os
user@host: /home$ print(os.environ["VARIABLE_NAME"})
```

11.3 PATH

PATH 환경 변수는 실행 프로그램에 대해 PATH 변수 내에 지정되어 있는 경로명을 생략하고 프로그램을 실행할 수 있게 한다. 예를들어 파이썬 인터프리터를 실행하고자 할때 /usr/bin/python2를 입력하는 것 대신에 python2를 입력하는 것만으로 가능하다. 즉. PATH 환경 변수는 저장소 안에 위치하는 실행 프로그램의 디렉토리를 저장함으로써 전체 경로를 알지 못하더라도 해당 프로그램을 실행할 수 있게끔 하는 변수이다.

```
user@host: /home$ export PATH="/usr/bin:$PATH"
```

11.4 Arguments of Programs

커맨드 라인 명령어와 동일하게 프로그램 또한 여러개의 옵션과 매개변수를 입력받을 수 있다. 입력 받는 매개변수의 개수는 매우 다양하며 short format과 longformat 모두 이용 가능하다.

12. Command Line and Python Scripts

12.1 Python Scripts

파이썬 스크립트(Python Script)란 파이썬 언어를 사용한 파일에서 여러 문장을 실행한 다음 멈추는 프로그램이다. 비록 셸 내부에서 python 명령어를통해 코드를 실행할 수 있지만, 어느 정도 규모가 있는 프로그램을 실행할 때 한줄씩 입력하는 방식은 효율적이지 못하다. 따라서 텍스트 파일이나 IDE통해 테스크 별 코드를 입력하고 한꺼번에 실행하는 방식이 효율적이다.

일반적으로 Jupyter, Pycharm, VSCode와 같은 Python IDE를 통해 개발하거나 text editor를 통해서 셸에서 실행하는 방식이 있다. 셸 내부에서 텍스트 파일을 생성하는 방법은 리다이렉션과 nano 명령어가 있다.

12.2 __main__

Python Script의 파일 내부에는 if **name** == "**main**" : 구문이 존재한다. Python Script가 다른 Script 내부에 임포트된 경우가 아니라 인터프리터에서 직접 실행된 경우에만 Code block 내부의 코드를 실행하라는 조건문이다.

12.3 Install packages

```
user@host: /home$ pip install [packages]
```

12.4 Virtual Environment

가상환경(Virtual Environment)은 하나의 PC에서 프로젝트 별로 독립된 파이썬 실행환경을 마련하고 독립된 패키지와 버전을 사용할 수 있도록 한다. 이를 통해 프로젝트 별로 다른 버전과 패키지를 설치하여 다른 프로젝트로부터 분리할 수 있기 때문에 시스템 전역 패키지 설치로 인한 불필요한 이슈를 방지할 수 있다.

```
# Install virtual environment
user@host: /home$ virtualenv -p /usr/bin/python3 virenv3

# Execute virtual environment
user@host: /home$ sourcec python3/bin/activate virenv3

# Check version of python
user@host: /home$ python -V

# Check installed packages in virtual environment
user@host: /home$ pip freeze

# Deactivate virtual environment
user@host: /home$ deacrivate
```

12.4 Import function to another script

한 스크립트 내부에서 생성한 함수는 다른 파이썬 스크립트로 호출이 가능하다. Python 패키지를 임포트하고 메소드를 사용하는 것과 동일하게, 해당 스크립트를 임포트 하고 내부의 함수를 메소드처럼 사용할 수 있다.

```
import script1

if __name__ == "__main__" :
    script1.func1()
```

12.5 Pass option into Python Script

명령어에 옵션을 추가하는 것과 마찬가지로 커맨드 라인에서 입력한 arguments를 파이썬 스크립트 내부에 전달하는 방법이 있다. Python의 sys 패키지의 argv 리스트는 내부에 positional argument로 입력한 값을 저장하고 있다.

```
user@host: /home$ python script.py "argument1"
```

13. Piping and Redirecting Ouput

13.1 Running two commands sequentially

두가지 이상의 명령을 동시에 수행해야 하는경우 && 연산자를 통해 서로 다른 명령을 수행할 수 있다.

```
user@host: /home$ echo "All the beers are gone" >> beer.txt && cat beer.txt
```

13.2 Using a backslash escape character

이전에 character escaper의 내용으로 " "와 "?" 과 같은 Wildcard로 인식될 수 있는 표현을 문자 그대로 인식해야 하는 경우 '\'를 통해 해당 문자를 그대로 인식시킬 수 있다.

```
user@host: /home$ echo "\"Get out of here.\"Said Neil Armstrong to the moon people." >> famous_quotes
```

14. Csvkit

CSVkit는 CSV파일을 다루고 처리하기 위한 명령언어 도구이다.

14.1 csvstack

csvstack은 복수의 파일에 존재하는 동일한 행을 열로 인식하고 이를 제외한 후 데이터를 합친다.

- -n : 새로운 열을 생성한다
- -g : 원래 파일을 구분할 수 있는 값을 저장한다

```
user@host: /home$ csvstack file1.csv file2.csv > newfile.csv
user@host: /home$ csvstack -n origin -g 1,2 file1.csv file2.csv > newfile.csv

# Challenges
user@host: /home$ head -n +1 Hud_2013.csv >> Combined_hud.csv
user@host: /home$ head -n +2 *.csv >> Combined_hud.csv

# Csvkit
user@host: /home$ csvstack -n year -g 2005,2007,2013 *.csv > Combined_hud.csv
```

14.2 csvlook

CSV파일은 기본적으로 tabular dataset이다. head 명령어는 separator에 의해 분리된 텍스트 파일을 출력하기 때문에 가독성이 떨어진다. csvlook 명령어는 CSV파일을 테이블 형태로 포맷해 데이터 테이블을 출력한다.

```
user@host: /home$ head -n file1.csv | csvlook
```

14.3 csvcut

csvcut 명령어는 cut 명령어와 비슷하게 CSV파일 내부의 특정 열을 추출하는데 사용된다.

- -n : 열이름을 정수로 추상화한다
- -c : 추상화된 정수열을 기준으로 데이터를 추출한다

```
user@host: /home$ csvcut -n Combined.csv
user@host: /home$ csvcut -c 1,5 Combined.csv | head -10
```

14.4 csvstat

csvstat 명령어는 csvcut을 통해 추출된 열에 대해서 단순한 기술통계량을 추출하는 명령어이다. --(max, min, ...) 옵션을 통해 특정 기술통계량을 검색할 수 있다.

```
user@host: /home$ csvstat --mean Combined_hud.csv
```

14.5 csvgrep

csvgrep 명령어는 grep 명령어와 동일하게 특정 열에서 패턴을 만족하는 모든 행을 출력한다.

```
user@host: /home$ csvgrep -[c] -[r] -[m][pattern] file1.csv
```