VIM & Shell Tools and Scripting

Taegyeong Lee / 2024.01.11

Lecture 1 Summary

The Shell

- **Shell** 이란 텍스트 인터페이스 환경으로 컴퓨터 명령을 입력, 실행하고 제어할 수 있게 해주는 것
- **Command-Line Shell (CLI)** : Bash, Command Prompt, PowerShell
- **Graphical Shell (GUI) :** 사용자가 편리하게 사용할 수 있게, 그래픽으로 나타낸 것
- 대부분의 서버는 우분투 환경
- **GUI를 지원하는 Vscode와 연동해서 코딩, 그러나 CLI 의 가장 기본적인 명령어는 필수로 알고 있어야 한다.**

Quiz 1

1. **tmp 폴더와 tmp폴더안에 test.txt 파일을 만들어보세요.**
2. **1번 문제를 |를 통해 한번에 해결해보세요. (;)**
3. **Vim 명령어를 통해 test.txt에 hello world를 저장해보세요.**
4. **Cat 명령어를 통해 test.txt 파일의 내용을 출력해보세요.**
5. Test2.txt를 만들고, test2에는 test.txt의 내용을 입력하세요. (**cat < hello.txt > hello2.txt)**
6. Tmp2 폴더를 만들고 tmp2폴더 안에 test.txt 파일을 만든다. 이때 test.txt에는 오늘 날짜가 입력되어있다. **배쉬 파일을 작성하고** 실행하시오. (Mkdir, touch, date)

Quiz 1 Plus

- 아래 두 명령어는 어떤 차이가 있을까 ?
- Ls | grep “.” > test1.txt ; cat < test1.txt | wc –l
- Ls | grep “.” > test1.txt | cat < test1.txt | wc -l

Ls | grep “.” > test1.txt ; cat < test1.txt | wc –l

- **Ls | grep “.” > test1.txt ; cat < test1.txt | wc –l**
- **Ls :** 현재 디렉토리의 파일과 디렉토리 목록을 출력
- **Grep “.”** 파일 및 디렉토리 목록에서 빈줄이 아닌 모든 줄 찾기
- **> test1.txt ..** 이 결과를 test1.txt에 저장
- **;** 연속적으로 다음 cat 명령 실행
- **Cat < …** Test1.txt를 읽어서 cat에 전달해서 출력
- **Wc –l Z**출력된 명령어에서 줄수를 세서 출력

Ls | grep “.” > test1.txt | cat < test1.txt | wc -l

- Ls | grep “.” > test1.txt f를 실행
- 출력한 결과를 |(파이프) 뒤의 명령으로 전달 따라서 cat <test1.txt | wc –l 부분은 실행 X

Solution 1

1. mkdir tmp, touch test.txt
2. Mkdir tmp;cd tmp;touch test.txt
3. vim test.txt
4. cat test.txt
5. cat < test.txt > test2.txt
6. mkdir -p /content/tmp2

cd /content/tmp2

touch /content/tmp2/test.txt

date > /content/tmp2/test.txt

Quiz 2

- 우분투의 폴더 안에서 파일 개수를 세는 명령어를 짜보시오
- (체인과 ls, grep, wc를 사용하시오)

Hint

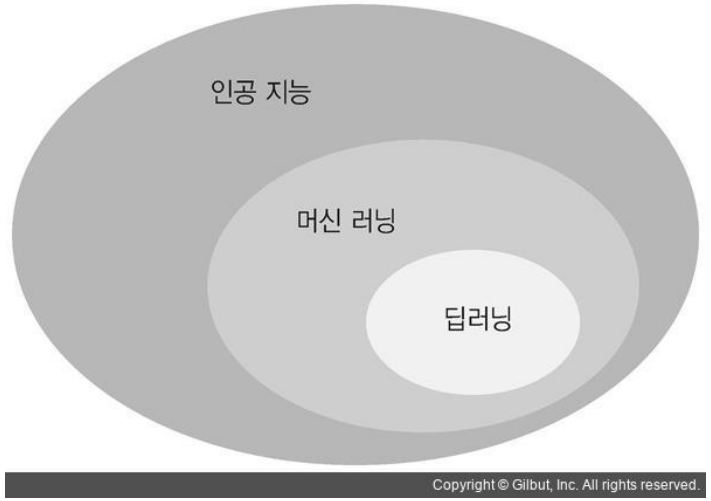
- ls와 grep, wc를 사용할 수 있다.
- grep ^-에서 ^-은 일반 파일들을 출력한다.
- wc –l 의 경우 word count로 단어를 분석하여 라인의 개수를 센다.
- wc myfile.txt -> 라인수, 단어수, 바이트수 출력

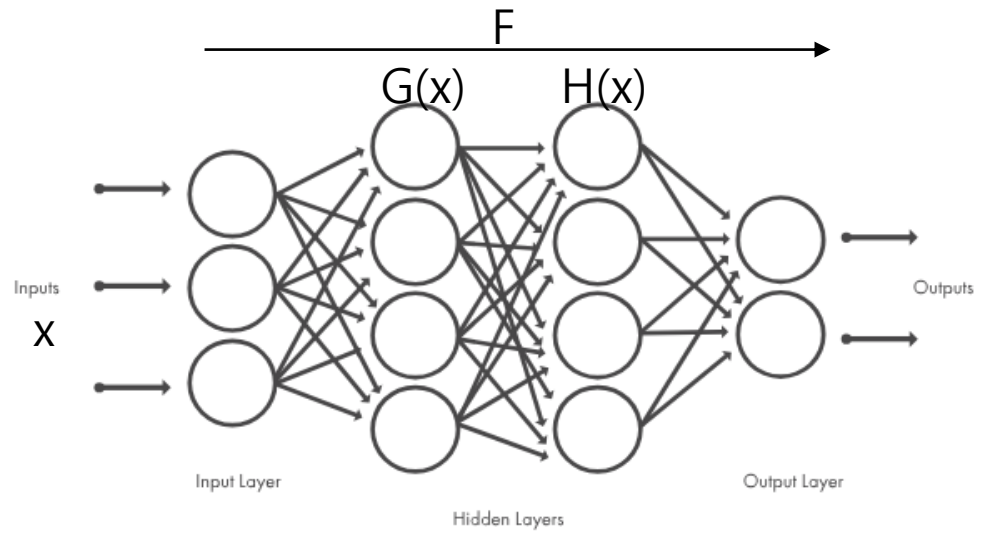
Solution 2

- **ls -l | grep ^- | wc –l**
- ls-l : 디렉토리 내용을 long format으로 나열
- Grep ^- ^- 텍스트 패턴이 –로 시작하는 경우 (-는 일반 파일), 일반파일 일 경우
- Wc –l : word count 텍스트 데이터 분석 –l은 라인 수

Deep Learning

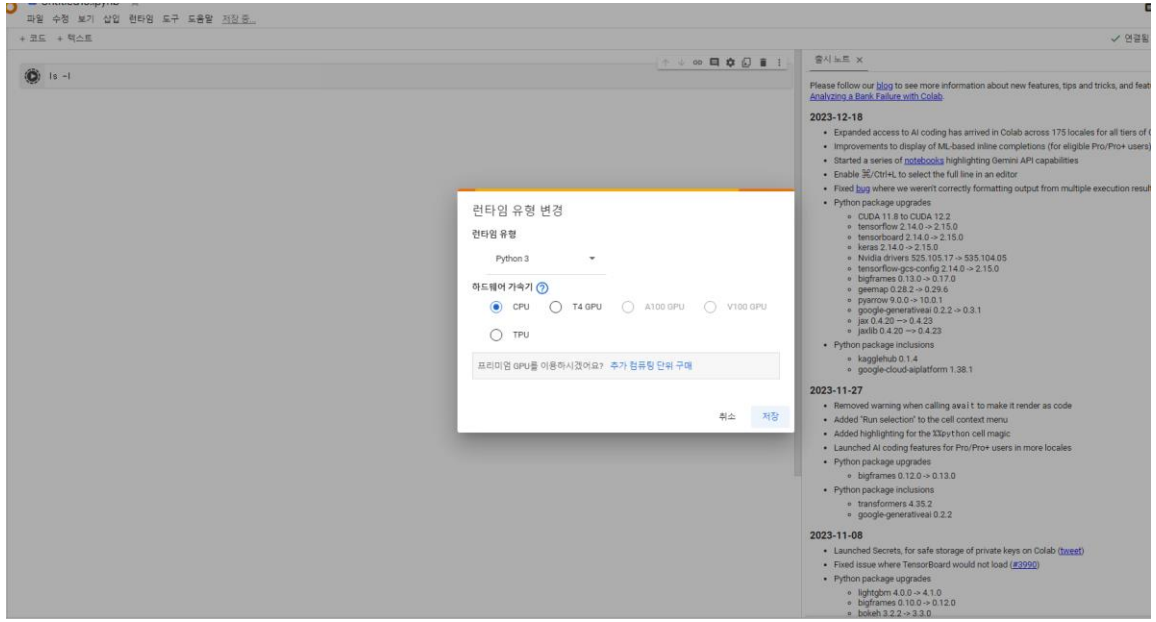
- Constitutes many layers to process incoming data
- **Function composition** / F = H(G(x)), 합성함수
- Fully-connected network (FC-net)
- Convolutional neural network (CNN, parameter sharing, local한 feature)
- Recurrent neural network (RNN, sequence data)





Colab 이란 ?

- 코랩(Colab)은 구글에서 제공하는 클라우드 기반의 Jupyter 노트북 환경입니다
- https://colab.research.google.com/
- GPU 사용하게 설정



파이썬 기초

- **print**(“출력할구문”)
- **print**(“a”*3) # a를 3번 출력
- **변수명** = “변수에 들어갈 내용”

- **리스트 타입** : []
- **딕셔너리(사전) 타입** : {}
- 사용자로부터 값을 입력받는 방법 input()

```
[1] print("test")

    a = "hello"
    print(a)
```

```
test
hello
```

```
my_list = []
my_dict = {}

my_dict['my_key'] = "my_value"
my_list.append(a)

print(my_dict)
print(my_list)
```

```
{'my_key': 'my_value'}
['hello']
```

파이썬 기초 (반복문)

- for 문, while 문
- for I in range(0,5):
- while 조건:
- for item in my_list:
- for idx , Item in enumerate(my_list):

파이썬 기초 (조건문)

- if 조건 :
- elif 조건:
- else:

Quiz

문제 1 (기본 for 문):
- 문제: 1부터 10까지의 숫자를 출력하는 파이썬 프로그램을 작성하세요.

문제 2 (while 문과 조건문):
- 문제: 사용자로부터 숫자를 계속 입력받다가, 0을 입력받으면 입력받은 숫자들의 평균을 출력하는 프로그램을 작성하세요.

문제 3 (리스트와 for 문):
- 문제: 주어진 리스트 **numbers = [1, 2, 3, 4, 5]**의 모든 요소에 10을 더한 새로운 리스트를 생성하는 파이썬 프로그램을 작성하세요.

문제 4 (enumerate 함수와 for 문):
- 문제: 리스트 **fruits = ['apple', 'banana', 'cherry']**의 각 요소와 해당 요소의 인덱스를 출력하는 파이썬 프로그램을 작성하세요.

문제 5
- 별(*) 패턴을 출력하는 파이썬 프로그램을 작성하세요. 출력 결과는 다음과 같아야 합니다:

```
*
**
***
****
*****
```

VIM & Shell Tools and Scripting

VIM

- **cat** 파일명 : 파일명에 대한 내용 출력
- **vim 파일명**
- **w 파일 저장, q vim 종료, wq 파일 저장 후 vim 종료, q! 변경 사항 저장x vim 강제종료**
- **ESC키 후, I 편집모드, a커서 위치 다음부터 편집, o 새로운 라인 추가 후 편집**

Shell Scripting

- 지난시간까지.. Shell에서 명령을 추출하고 파이프로 연결
- But, 대부분의 상황은 “조건문”, “반복문“ 등이 필요
- 우리는 가장 일반적인 bash scripting에 중점을 둘것

Shell Scripting Variable

아래 코드를 입력하고 결과를 살펴보자.

foo=bar ? Foo = bar

Echo “$foo” -> bar 출력 (변수의 값 출력)
Echo ‘$foo’ -> $foo 출력 (문자열로 해석)
Echo $foo -> bar 출력 (변수의 값 출력)

Shell Scripting Function

```
mcd () {
mkdir –p “$1”
cd “$1”
}
```

- `$0` - 스크립트 이름
- `$1` ~ `$9` - 스크립트의 인자들. `$1` 부터 첫번째 인자입니다.
- `$@` - 모든 인자들
- `$#` - 인자의 수
- `$?` - 이전 명령 수행결과 코드 (error code). C 프로그래밍 언어와 같이 0은 정상적으로 종료되었음을 의미합니다.
- `$$` - 현재 스크립트에 대한 프로세스 식별 번호 (PID)
- `!!` - 인수를 포함하여 마지막 명령 전체를 포함합니다. 일반적으로는 사용 권한이 누락되어 실패할 때 사용합니다. `sudo` 를 함께 써서 실패한 명령을 신속하게 다시 실행할 수 있습니다.
- `$_` - 마지막 명령에서 나온 마지막 인수입니다. 대화형 셸에 있는 경우 `Esc` 를 입력한 후 `.` 을 입력해 이 값을 신속하게 얻을 수 있습니다.

Shell Scripting

false || echo "Oops, fail" -> False는 1을 리턴, **||는 Or 명령어로 왼쪽 실패시 오른쪽 명령어 실행**
Oops, fail

true || echo "Will not be printed" **-> True는 0을 리턴**
#

true && echo "Things went well" **-> and 명령어로 둘다 실행**
Things went well

false && echo "Will not be printed" **-> and 명령어로 실패했음으로 실행 X**
#

true ; echo "This will always run" **-> 연속 실행**
This will always run

false ; echo "This will always run" **-> 연속 실행**
This will always run

Shell Scripting 연산자

-eq: equal (같음)

-ne: not equal (다름)

-gt: greater than (보다 큼)

-ge: greater than or equal to (보다 크거나 같음)

-lt: less than (보다 작음)

-le: less than or equal to (보다 작거나 같음)

Shell Scripting for문

for 변수 in 범위
Do
	# 명령
Done

list = “A B C D E F G”
for var in $list
Do
	Echo $var
done

Shell Scripting while 문

```
while true
do
        echo “무한루프”
        sleep 1
done

count=1
while [ $count -le 5 ]  # less than or equal to
do
  echo "루프 실행 횟수: $count"
  count=$((count + 1))
done
```

Shell Scripting 조건문

a = "abc"

B = "abc"

If [$a –eq $b]; then

    echo yes

else

    echo no

Fi

If [1 –eq 1]

If ["abc" == "abc"]

If ['ls | wc =l' == 1]

Shell Scripting Wildcard

- **Wildcards - ?** 및 *을 사용하여 하나 또는 원하는 양의 문자와 일치
- Foo1, foo2, foo10, bar
- rm foo? Foo1과 foo2 삭제, rm foo * bar를 제외하고 삭제

Quiz 1

카운트다운 스크립트: (while 문 활용)

- 사용자에게 숫자를 입력받습니다.
- 입력받은 숫자부터 1까지 역순으로 카운트다운을 출력합니다.
- 각 숫자는 1초 간격으로 출력합니다.

구구단 출력 스크립트: (for문 활용)

- 사용자에게 2에서 9 사이의 숫자를 입력받습니다.
- 입력받은 숫자에 해당하는 구구단을 출력합니다.
- 예: 사용자가 2를 입력하면, 2단 전체를 출력합니다.

```
#!/bin/bash
read number
count=$((count + 1))
```

Shell Scripting

```
#!/bin/bash -> 스크립트 파일이 Bash 쉘을 사용하여 실행해야한다.

echo "Starting program at $(date)" # Starting program과 오늘날짜 출력
echo "Running program $0 with $# arguments with pid $$"
# $0 : 실행중인 프로그램의 이름, $# 전달받은 인자의 개수, 현재 프로세스의 아이디 $$ 를 출력

for file in $@; do  # for문 시작
    grep foobar $file > /dev/null 2> /dev/null
#>dev/null 2 > /dev/null 표준출력 및 표준 에러 모두 무시

    # When pattern is not found, grep has exit status 1
    # We redirect STDOUT and STDERR to a null register since we do not care about them
    if [[ $? -ne 0 ]]; then  # 마지막에 실행된 명령의 종료상태 "$?"를 확인. Grep은 패턴을 찾지 못하면 종료상태는 1이 됨. –ne(not equal) 0은 종료상태가 0인지 아닌지를 확인
        echo "File $file does not have any foobar, adding one"
        echo "# foobar" >> "$file "  #파일에 foobar가 없을경우, $file끝에 foobar를 추가
    fi
done
```

Shell Scripting

Find all directories named src
find . -name src -type d
Find all python files that have a folder named test in their path
find . -path '/test/**/*.py' -type f**
Find all files modified in the last day
find . -mtime -1
Find all zip files with size in range 500k to 10M
find . -size +500k -size -10M -name '*.tar.gz'

Delete all files with .tmp extension
find . -name '*.tmp' -exec rm {} ₩;
Find all PNG files and convert them to JPG
find . -name '*.png' -exec convert {} {.}.jpg ₩;

Quiz 2

1. 파일 이름 검사 스크립트:

사용자로부터 파일 이름을 입력받습니다.

해당 파일이 디렉토리에 존재하는지 확인합니다.

파일이 존재하면 "파일이 존재합니다."를 출력하고, 존재하지 않으면 "파일이 존재하지 않습니다."를 출력합니다.

2. 숫자 맞추기 게임 스크립트:

스크립트가 1부터 100 사이의 랜덤한 숫자를 생성합니다.

사용자에게 숫자를 추측하도록 요청합니다.

사용자의 추측이 정답보다 낮으면 "더 높은 숫자입니다."를, 높으면 "더 낮은 숫자입니다."를 출력합니다.

사용자가 정답을 맞추면 "정답입니다!"를 출력하고 스크립트를 종료합니다.

Thank you