Shell Tools and Scripting

A Basic CS Skill, ABC Winter School

박원

Objective

- Shell Scripting을 배울 거에요

Simple Vim

```
[abc@unist ~]$ vim <file_name>
```

- 위의 명령어는
  - <file_name>이 없으면 새로 만들고,
  - 있다면 이미 있는 파일을 수정해요.

Simple Vim

- vim 화면에 처음 들어가면, i를 눌러서 수정모드에 진입할 수 있어요.
  - i를 누르면 좌하단에 `-- INSERT --` 라는 문구가 나오며 수정모드라는 것을 알 수 있어요.
- 수정이 끝났다면 `esc`를 가볍게 한번 눌러줍시다.
- `:wq`를 입력하면 저장 후 종료가 되어요.
- `:w`는 그냥 저장, `:q`는 그냥 종료이고, 모든 명령어 뒤에 `!`를 붙이면 강제로 실행해요.

Simple Vim

- 수정이 잘 되었나 확인해볼까요?

```
[abc@unist ~]$ vim hello.txt
[abc@unist ~]$ cat hello.txt
hello!
```

- `echo`랑 달리 `cat`은 파일의 내용을 읽어와서 출력합니다.

Shell Scripting?

- 모든 작업을 할 때, shell 상에서 자동화가 필요한 경우가 있습니다.
- 간단한 연산자와 옵션들 만으로, 프로토타입을 간단히 만들 수 있어요.
  - 테스트 스크립트도 작성이 가능합니다!

Shell Scripting - More about `echo`

```
echo "Hello" > hello.txt
```

vs

```
echo "Hello!" >> hello.txt
```

- 전자는 파일을 새로 쓰기, 후자는 이어쓰기

Shell - More Useful Commands

cat <target_file>

- `<target_file>`의 내용을 읽어와요!

diff <file1> <file2>

- 두 파일의 내용을 비교해요.
  `diff <(command) <(command)`
- 이런식으로 명령어의 출력도 비교할 수 있어요!

Shell - More Useful Commands

touch

- target file의 최종 수정 일자를 최신으로 만들어줍니다.
- 파일이 없다면 빈 파일을 새로 만들어요!

```
mkdir foo bar
touch {foo bar}/{a..h}
touch foo/x foo/y
diff <(ls foo) <(ls bar)
```

Shell Scripting - input

```
python3 test.py < input.txt
```

```
# test.py
a = int(input())
print(a*2)
```

Shell Scripting - Variable

```
# variable.sh
foo=bar
echo $foo
echo "$foo"
echo '$foo'
```

```
[abc@unist ~]$ ./variable.sh
bar
bar
$foo
```

- zsh에서 bash script를 실행할땐 `bash ./variable.sh` !!

Shell Scripting - Special Variables

Variable	Value
$0	Name of the script
$1 - $9	Arguments of script. $1 is the first
$@	All arguments
$#	Number of arguments
$?	Return code of the previous command
$$	Process identification number(PID) for current script
!!	Entire last command
$_	Last argument from the last command

Shell Scripting - Truth Value

```
false || echo "False!!"
# False!!
true || echo "Won't be printed"
#
true && echo "True!!"
# True!!
false && echo "Won't be printed"
#
true ; echo "Always run"
# Always run
false ; echo "Always run"
# Always run
```

Shell Scripting - Array

- 배열도 선언할 수 있어요!

```
array = (1 2 3)
array = ("v1" "v2" "v3)
```

- python, c는 `,`로 구분하지만, 여기는 공백으로 구분합니다.
- 배열 접근은 다음과 같이 합니다.

```
${array[1]}
${array[$i]}
```

Shell Scripting - Arithmetic

- shell 스크립트는 기본적으로 모두 문자로 취급해요.
  - 숫자로 취급해서 산술 연산을 하려면 어떻게 해야할까요?
- 아래처럼 사용합니다.

  `$((expression))`

```
a = $((4 + 5))
b = $(($1 + $2))
```

Shell Scripting - If/Else

```
if [[ condition ]]
then
    command
else
    command
fi
```

Shell Scripting - If/Else

- Example

```
# pos-neg.sh
if $1 > 0
then
    echo "Positive!"
else
    echo "Negetive or zero!"
fi
```

Shell Scripting - For loop

```
for day in Mon Tue Wed Thur Fri Sat Sun
do
    echo $day
done

for day in "Mon Tue Wed Thur Fri Sat Sun"
do
    echo $day
done
```

Shell Scripting - For Loop (C Style)

```
for ((a=1; a <= 10; a++))
do
    echo $a
done
```

실습 1 - 큰 수 출력하기

- 두 수를 입력 받아서, 둘 중 더 큰 수를 출력하는 스크립트를 만들어 봅시다.

```
[abc@unist ~]$ ex1.sh 10 9
10
[abc@unist ~]$ ex1.sh 8 12
12
```

실습 2 - 수 출력하기

- 두 수 a, b를 입력받아서 a 이상 b 이하의 모든 수를 출력해보아요.
- a가 b보다 큰 경우는 무시할게요.

```
[abc@unist ~]$ ex2.sh 1 5
1
2
3
4
5
```

실습 3 - 배수 출력하기

- 세 수 a, b, c를 순서대로 입력받아서, a 이상 b 이하의 수 중 c의 배수만 출력해보아요!
- 힌트: 배수인지 확인하려면 어떤 수 `$i`에 대해서 `$(($i % $3))`이 0인지 확인하면 돼요!

```
[abc@unist ~]$ ex3.sh 1 10 3
3
6
9
```

실습 4 - 파이썬 테스트 코드 짜기

- 파이썬 코드를 짜고, 파이썬 코드를 테스트 해볼게요.
- 같이 해볼게요!

Questions?

Next