Shell Scripts

2021 Winter Wheel Seminar

2021/01/07

paco

샘플 코드 주소

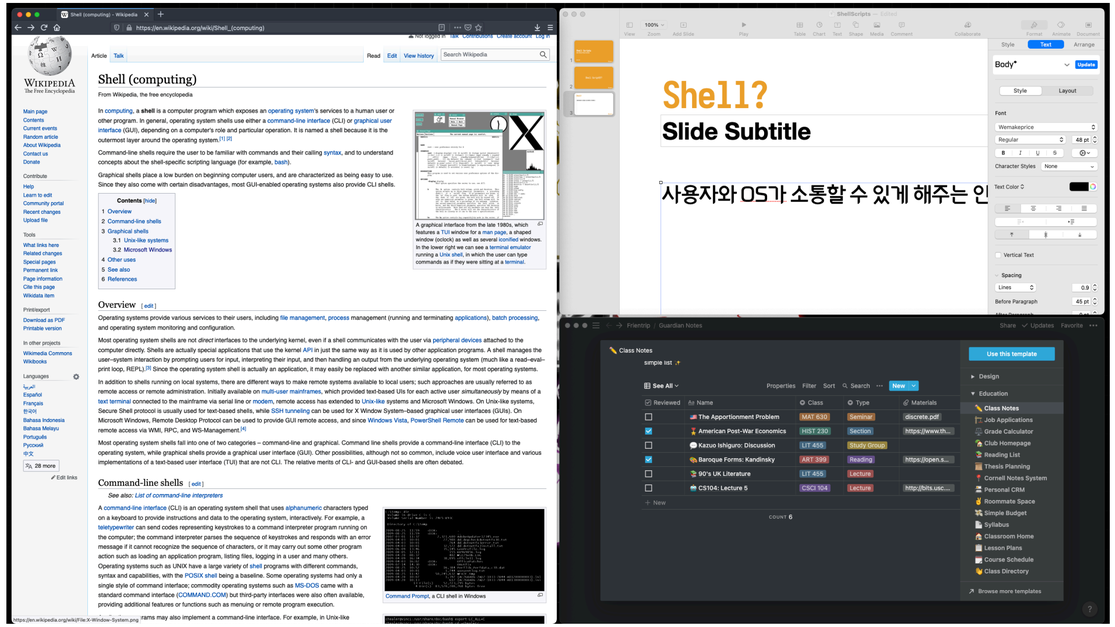
paco

Shell Script란?

Shell?

사용자와 OS가 소통할 수 있게 해주는 인터페이스

Command-line Shell



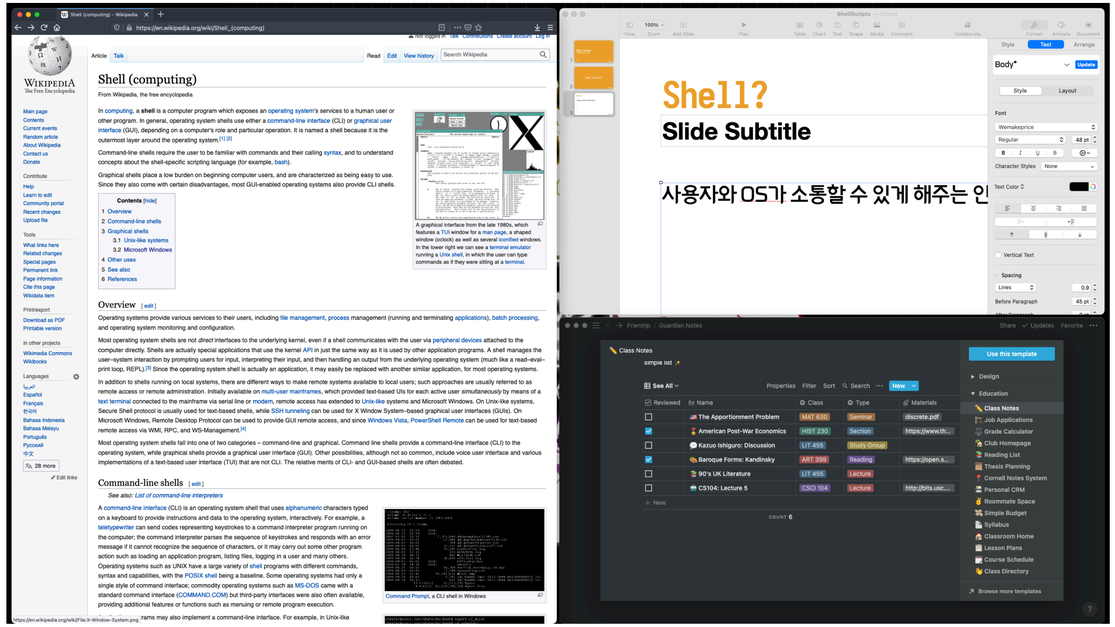
Graphical Shell

paco

Shell?

사용자와 OS가 소통할 수 있게 해주는 인터페이스

Command-line Shell



Graphical Shell

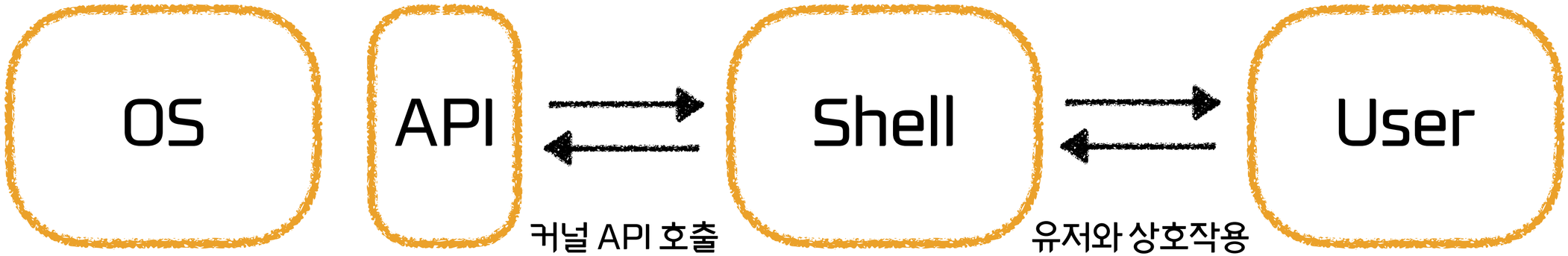
오늘은 이것에 집중할 것

paco

Shell?

셸도 결국은 프로그램

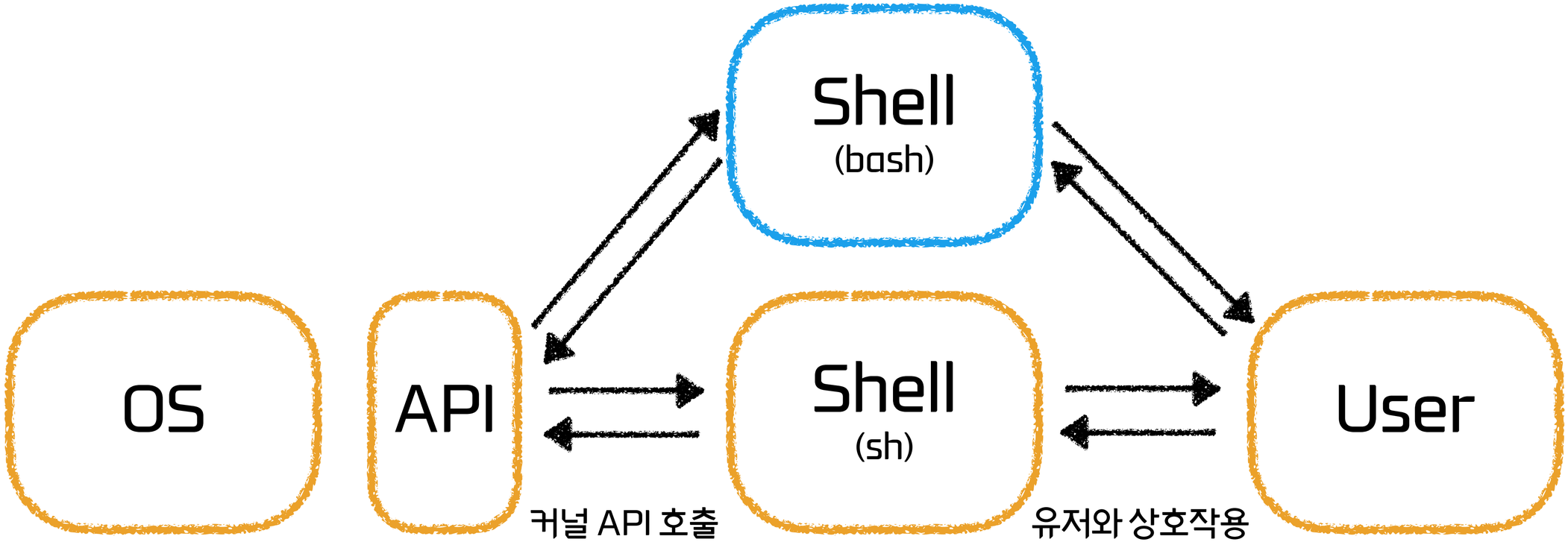
- 파일 관리
- 프로세스 관리
- 등등..



paco

Shell?

셸도 결국은 프로그램



- 파일 관리
- 프로세스 관리
- 등등..

paco

Shell Script? : 셸을 이용해 실행할 수 있는 프로그램

paco

Shell Script? : 셸을 이용해 실행할 수 있는 프로그램

어떤 언어를 사용해서 작성하나요?

Shell Script? : 셸을 이용해 실행할 수 있는 프로그램

어떤 언어를 사용해서 작성하나요?

bash (에서 사용하는 언어) 기준으로!

Shell == Interpreter

코드를 컴파일하지 않고 줄 단위로 해석 / 실행함

```
❯ echo "foo"
foo
❯ ls /Users
Shared    pacokwon
❯ python -c "print('Wheel Seminar')"
Wheel Seminar

~ ❯                                          18:43:10
```


paco

Shell == Interpreter

코드를 컴파일하지 않고 줄 단위로 해석 / 실행함

```
❯ echo "foo"
foo
❯ ls /Users
Shared    pacokwon
❯ python -c "print('Wheel Seminar')"
Wheel Seminar

~ ❯                                          18:43:10
```

오늘의 Objective: 커맨드들을 조합하여 프로그램을 짠 뒤 셸을 이용해 실행해보기!

paco

파일 생성과 실행

```
~/w/shellscript ❯ vim test.sh                    20:18:31
```

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
```

```
❯ bash test.sh
Hello, SPARCS!

~/w/shellscript ❯                                20:18:31
```


paco

shebang (#!)

스크립트 맨 윗줄에 interpreter를 명시

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
```

#! 뒤에 있는 경로의 interpreter를 이용해 파일 실행이 가능!

```
❯ ./test.sh
zsh: permission denied: ./test.sh
❯ ls -l test.sh
-rw-r--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ chmod u+x test.sh
❯ ls -l test.sh
-rwxr--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ ./test.sh
Hello, SPARCS!
```

(execution permission 부여 이후 실행)

shebang (#!)

스크립트 맨 윗줄에 interpreter를 명시

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
```

#! 뒤에 있는 경로의 interpreter를 이용해 파일 실행이 가능!

```
❯ ./test.sh
zsh: permission denied: ./test.sh
❯ ls -l test.sh
-rw-r--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ chmod u+x test.sh
❯ ls -l test.sh
-rwxr--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ ./test.sh
Hello, SPARCS!
```

(execution permission 부여 이후 실행)

Q: 왜 절대경로를 적어주어야 하나?

A: shebang handling은 커널이 하는데, 커널은 (PATH를 비롯한) 환경변수를 해석하지 않기 때문에 interpreter를 잘 찾아갈 수 있게 경로를 명시해야 한다. (따라서 상대경로도 가능은 함)

paco

shebang (#!)

스크립트 맨 윗줄에 interpreter를 명시

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
```

#! 뒤에 있는 경로의 interpreter를 이용해 파일 실행이 가능!

```
❯ ./test.sh
zsh: permission denied: ./test.sh
❯ ls -l test.sh
-rw-r--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ chmod u+x test.sh
❯ ls -l test.sh
-rwxr--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ ./test.sh
Hello, SPARCS!
```

(execution permission 부여 이후 실행)

Q: 왜 절대경로를 적어주어야 하나?

A: shebang handling은 커널이 하는데, 커널은 (PATH를 비롯한) 환경변수를 해석하지 않기 때문에 interpreter를 잘 찾아갈 수 있게 경로를 명시해야 한다. (따라서 상대경로도 가능은 함)

Q: 왜 실행할 때 test.sh와 같은 식으로는 실행이 안되는가?

A: PWD(현재 경로)는 PATH에 없기 때문.

paco

shebang (#!)

스크립트 맨 윗줄에 interpreter를 명시

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
```

#! 뒤에 있는 경로의 interpreter를 이용해 파일 실행이 가능!

```
❯ ./test.sh
zsh: permission denied: ./test.sh
❯ ls -l test.sh
-rw-r--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ chmod u+x test.sh
❯ ls -l test.sh
-rwxr--r--  1 pacokwon  staff  34 Jan  3 20:18 test.sh
❯ ./test.sh
Hello, SPARCS!
```

(execution permission 부여 이후 실행)

Q: 왜 절대경로를 적어주어야 하나?

A: shebang handling은 커널이 하는데, 커널은 (PATH를 비롯한) 환경변수를 해석하지 않기 때문에 interpreter를 잘 찾아갈 수 있게 경로를 명시해야 한다. (따라서 상대경로도 가능은 함)

Q: 왜 실행할 때 test.sh와 같은 식으로는 실행이 안되는가?

A: PWD(현재 경로)는 PATH에 없기 때문.

Q: 시스템마다 bash가 다른 곳에 위치해 있으면 어떡하나?

A: 이식성(portability)과 관련된 문제. env command를 사용한다.

paco

shebang (#!)

Q: 시스템마다 bash가 다른 곳에 위치해 있으면 어떡하나?

A: 이식성(portability)과 관련된 문제. env command를 사용한다.

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
~
~
~
"test.sh" 3L, 35C written
```

```
#!/usr/bin/env bash
echo "Hello, SPARCS!"
~
~
~
~
~
"test.sh" 3L, 43C written
```


paco

shebang (#!)

Q: 시스템마다 bash가 다른 곳에 위치해 있으면 어떡하나?

A: 이식성(portability)과 관련된 문제. env command를 사용한다.

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
~
~
~
"test.sh" 3L, 35C written
```

Q: 시스템마다 env가 다른 곳에 위치해 있으면 어떡하나?

```
#!/usr/bin/env bash
echo "Hello, SPARCS!"
~
~
~
~
~
"test.sh" 3L, 43C written
```


paco

shebang (#!)

Q: 시스템마다 bash가 다른 곳에 위치해 있으면 어떡하나?

A: 이식성(portability)과 관련된 문제. env command를 사용한다.

```
#!/bin/bash
echo "Hello, SPARCS!"
~
~
~
~
~
"test.sh" 3L, 35C written
```

Q: 시스템마다 env가 다른 곳에 위치해 있으면 어떡하나?

A: 충분히 가능한 시나리오 (제재할 만한 규약이 없음). 그러나 env는 /usr/bin 에 위치한 경우가 잦다고 한다.

```
#!/usr/bin/env bash
echo "Hello, SPARCS!"
~
~
~
~
~
"test.sh" 3L, 43C written
```


paco

Variables 변수

Variables 변수

VARIABLE=VALUE

로 변수 생성 가능. 공백 두면 안됨!

변수들은 값을 문자열로 담는다.

paco

greet 프로그램 작성하기

greet – 이름을 input으로 받은 뒤 인사말 출력

(샘플 코드 greet 참고)

`read name`: name이라는 변수에 에 입력값 저장

`“Hello, $name”`: 문자열 안에 변수 값 끼워넣기

paco

greet 프로그램 작성하기

greet – 이름을 input으로 받은 뒤 인사말 출력

(샘플 코드 greet 참고)

`read name`: name이라는 변수에 에 입력값 저장

`“Hello, $name”`: 문자열 안에 변수 값 끼워넣기

```
❯ name=paco
❯ echo "$name_file"

❯ echo "${name}_file"
paco_file
```


paco

Variable Scope

test

```
  2 #!/usr/bin/env bash
  1
3   echo "foo is: $foo"
  1 foo="some string"
  2 echo "foo is: $foo"
~
~
~
~
```

Q: 변수를 미리 초기화하고 실행하면??

```
❯ foo=FOO
❯ ./test
```


paco

Variable Scope

test

```
2 #!/usr/bin/env bash
1
3 echo "foo is: $foo"
1 foo="some string"
2 echo "foo is: $foo"
~
~
~
~
```

Q: 변수를 미리 초기화하고 실행하면??

```
❯ foo=FOO
❯ ./test
foo is:
foo is: some string
```

A: 안 나옴. 나오게 하기 위해서는 export해야 한다

export하면 환경변수가 된다!

paco

Variable Scope

```
test
  2 #!/usr/bin/env bash
  1
3   echo "foo is: $foo"
  1 foo="some string"
  2 echo "foo is: $foo"
~
~
~
~
```

Q: 변수를 미리 초기화하고 실행하면??

```
> foo=FOO
> ./test
foo is:
foo is: some string
```

A: 안 나옴. 나오게 하기 위해서는 export해야 한다

export하면 환경변수가 된다!

```
> export foo=FOO
> ./test
foo is: FOO
foo is: some string
```


paco

Variable Scope

```
test
  2 #!/usr/bin/env bash
  1
3   echo "foo is: $foo"
  1 foo="some string"
  2 echo "foo is: $foo"
~
~
~
~
```

(샘플 코드 export 참고)

Q: 스크립트에서 환경변수를 수정한다면?

```
❯ export foo=FOO
❯ ./test
foo is: FOO
foo is: some string
❯ echo $foo
FOO
```

```
❯ export foo=FOO
❯ source test
foo is: FOO
foo is: some string
❯ echo $foo
some string
```

A: 변경사항이 수정되지 않는다!

수정사항을 그대로 반영하고 싶으면, 해당 파일을 source하면 됨

Command Substitution with $(…)

```
❯ myname=`whoami`
❯ echo $myname
pacokwon
```

```
❯ myeditor=$(echo $EDITOR)
❯ echo $myeditor
/usr/local/bin/nvim
```

Bash performs the expansion by executing command in a **subshell** environment and replacing the command substitution with the standard output of the command, with any trailing newlines deleted
– man bash –

paco

Special Variables

- $0: 현재 스크립트의 파일명
- $1 ~ $n: 셸에 전달된 argument 값들 (“positional parameters”)
- $#: 셸에 전달된 argument 개수
- $?: 직전 명령어의 exit status
- $$: 셸의 PID
- $@: argument의 리스트
- $*: argument의 리스트

Special Variables

- $0: 현재 스크립트의 파일명
- $1 ~ $n: 셸에 전달된 argument 값들 (“positional parameters”)
- $#: 셸에 전달된 argument 개수
- $?: 직전 명령어의 exit status
- $$: 셸의 PID
- $@: argument의 리스트
- $*: argument의 리스트

```
./script first second third fourth
$0       $1    $2     $3    $4
```


paco

Special Variables

- $0: 현재 스크립트의 파일명
- $1 ~ $n: 셸에 전달된 argument 값들 (“positional parameters”)
- $#: 셸에 전달된 argument 개수
- $?: 직전 명령어의 exit status
- $$: 셸의 PID
- $@: argument의 리스트
- $*: argument의 리스트

```
“$*” : all positional parameters as a single word
“$@” : all positional parameters as separate strings
$@, $*: both subject to word splitting
```


paco

Special Variables

- $0: 현재 스크립트의 파일명
- $1 ~ $n: 셸에 전달된 argument 값들 (“positional parameters”)
- $#: 셸에 전달된 argument 개수
- $?: 직전 명령어의 exit status
- $$: 셸의 PID
- $@: argument의 리스트
- $*: argument의 리스트

```
“$*” : all positional parameters as a single word
“$@” : all positional parameters as separate strings
$@, $*: both subject to word splitting
```

(샘플 코드 at-asterisk 참고)

paco

Escaping Characters

특정 문자들은 문자열 내에서 escape되어야 올바르게 표시된다.

```
❯ echo "Hello \"World\""
Hello "World"
❯ echo "Hello "World""
Hello World
```

```
❯ foo=HELLO
❯ echo "\$foo"
$foo
❯ echo "$foo"
HELLO
```

```
bash-5.1$ echo "\\"
\
bash-5.1$ echo "\\\\"
\\
```

실습

greet 프로그램을 수정하여 마지막 출력 메시지에
hostname이 담기도록 해보자!

```
❯ ./greet
What is your name? paco
Hello, paco. You are on Pacos-Macbook-Pro.local
```

Operators 연산자

Integer Operators

- `expr $a <operator> $b`
- `$(($a <operator> $b))`

후자의 경우는 변수 앞 $ 생략 가능.

후자의 경우는 $((expression))의 골임

```
❯ a=3
❯ b=5
❯ expr $a + $b
8
❯ echo $((a + b))
8
```

이용 가능한 연산자들

- `+, -, *, /, %, +=, -=, *=, /=, %=, **=`
- `++ (pre, post), — (pre, post)`
- `&, |, ^, ~, <<, >>, &=, |=, ^=, <<=, >>=`

Floating Point

bash에서 자체적으로는 지원 안함. 그러나 외부 툴들을 이용해 할 수는 있다.

ex> Basic Calculator

```
❯ echo 'scale=4;20+5/2' | bc
22.5000
❯ echo 'scale=4;20+5/3' | bc
21.6666
❯ echo 'scale=5;20+5/3' | bc
21.66666
❯ echo $((20+5/3))
21
```

Relational Operators

for integers values

문법	의미
`[ $A -gt $B ]`	A > B
`[ $A -lt $B ]`	A < B
`[ $A -ge $B ]`	A ≥ B
`[ $A -le $B ]`	A ≤ B
`[ $A -eq $B ]`	A == B
`[ $A -ne $B ]`	A ≠ B

Relational Operators

for string values

문법	의미
[$A = $B]	A == B (문법에서는 = 한 개 주의)
[$A ≠ $B]	A ≠ B
[$A \> $B]	A is alphabetically greater than B
[$A \< $B]	A is alphabetically lesser than B
[-z $A]	A's length is 0
[-n $A]	A's length is not 0

Logical Operators

문법	의미
`command1 && command2`	command1의 exit status가 0일 때만 command2가 실행됨
`command1 || command2`	command1의 exit status가 1일 때만 command2가 실행됨
`command1; command2`	그냥 둘다 실행

Logical Operators

command1	가 0일 때만
command1	가 1일 때만
command1	

```
> true && echo hello!
hello!
> false && echo hello!
> true || echo hello!
> false || echo hello!
hello!
```


paco

Relational Operators

for integers values

문법	의미
`[ $A -gt $B ]`	A > B
`[ $A -lt $B ]`	A < B
`[ $A -ge $B ]`	A ≥ B
`[ $A -le $B ]`	A ≤ B
`[ $A -eq $B ]`	A == B
`[ $A -ne $B ]`	A ≠ B

Relational Operators

for integers values

문법	의미
[$A -gt $B]	A > B
[$A -lt $B]	A < B
	A ≥ B
	A ≤ B
	A == B

```
❯ a=3;b=5
❯ [ $a -lt $b ] && echo "$a is lesser than $b"
3 is lesser than 5
❯ [ $a -gt $b ] && echo "$a is lesser than $b"

~ ❯                                          01:07:16
```


paco

실습

number라는 변수에 정수가 저장되어 있다고 가정을 할 때,
number가 양수인지 여부를 판단해주는 스크립트를 작성해보자

```
❯ export a=10
❯ ./is-positive
a is positive!
❯ a=-1
❯ ./is-positive
a is not positive!
```

Conditionals 조건문

if statement

```
if [ expression ]
then
  commands
elif [ expression ]
then
  commands
else
  commands
fi
```

if statement

```
if [ expression ]
then
   commands
elif [ expression ]
then
   commands
else
   commands
fi
```

```
#!/usr/bin/env bash

a=3
b=5
if [ $a -lt $b ]
then
    echo $a is lesser than $b
else
    echo $a is greater than or equal to $b
fi
```

(샘플 코드 if 참고)

case statement

```
case [ expression ] in
PATTERN_1)
  commands
  ;;
PATTERN_2)
  commands
  ;;
PATTERN_3)
  commands
  ;;
*)
  commands
  ;;
fi
```

case statement

```
case [ expression ] in
PATTERN_1)
   commands
   ;;
PATTERN_2)
   commands
   ;;
PATTERN_3)
   commands
   ;;
*)
   commands
   ;;
fi
```

case

```
#!/usr/bin/env bash

echo -n "Enter a country name: "
read country

echo -n "$country's capital is "
case $country in
    [kK]orea)
        echo Seoul
        ;;
    [jJ]apan)
        echo Tokyo
        ;;
    [cC]hina)
        echo Beijing
        ;;
    *)
        echo "... sorry I don't know"
        ;;
esac
```

(샘플 코드 case 참고)

https://www.gnu.org/software/bash/manual/html_node/Pattern-Matching.html#Pattern-Matching

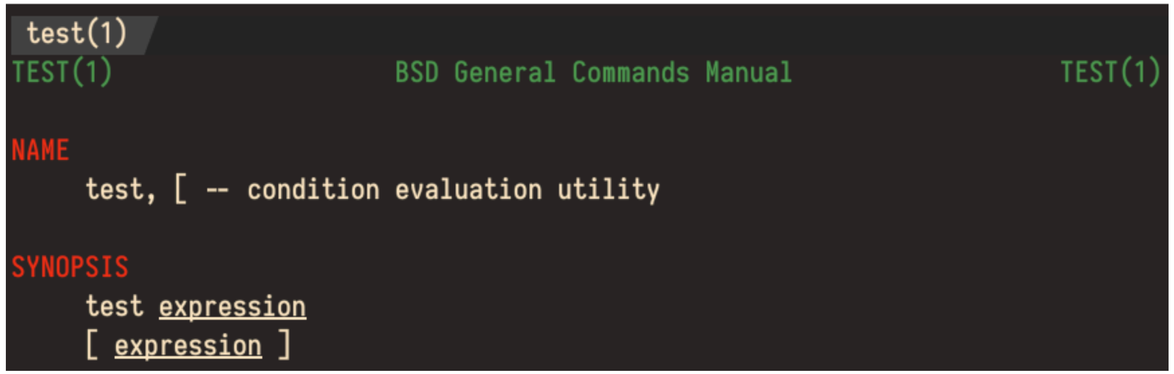
paco

Relational Operators

[의 정체는 무엇일까

Relational Operators

[의 정체는 무엇일까



paco

Relational Operators

[[는 뭘까?

- [is POSIX
- [[is bash extension

[[	[(POSIX equivalent)
`[[ a < b ]]`	`[ a \< b ]`
`[[ a = a && b = b ]]`	`[ a = a ] && [ b = b ]`
`[[ (a = a || a = b) && a = b ]]`	`{ [ a = a ] || [ a = b ]; } && [ a = b ]`
`x='a b'; [[ $x = 'a b' ]]`	`x='a b'; [ "$x" = 'a b' ]`

paco

실습

파일명을 인자로 받은 뒤 해당 파일의 존재 여부, 만약 존재한다면 regular 파일인지 디렉토리 인지 , 혹은 이외의 파일인지 출력하는 여부를 출력하는 스크립트를 작성해보자
(man test 참고)

```
❯ ./query-file /dev/stdout
/dev/stdout is some other file.
❯ ./query-file function
function is a regular file.
❯ ./query-file .
. is a directory.
❯ ./query-file hello
hello does not exist.
```

Loops 반복문

For statement

```
for i in var1 var2 var3 ...
do
  commands
done
```

```
for ((i=0;i<100;i++))
do
  commands
done
```

Expansion

: 유저가 입력한 커맨드에서 최종 명령문을 만들기 전에 이루어지는 일종의 가공 과정. 종류가 무려 7개

- Brace Expansion
- Tilde Expansion
- Parameter Expansion
- Command Substitution
- Arithmetic Expansion
- Word Splitting
- Pathname Expansion

Expansion

: 유저가 입력한 커맨드에서 최종 명령문을 만들기 전에 이루어지는 일종의 가공 과정. 종류가 무려 7개

- Brace Expansion
- Tilde Expansion
- Parameter Expansion
- Command Substitution
- Arithmetic Expansion
- Word Splitting
- Pathname Expansion

```
❯ echo {1..9}
1 2 3 4 5 6 7 8 9
❯ echo {a..z}
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

```
❯ touch someFile.{c,h}
❯ ls
someFile.c someFile.h
```

Expansion

: 유저가 입력한 커맨드에서 최종 명령문을 만들기 전에 이루어지는 일종의 가공 과정. 종류가 무려 7개

- Brace Expansion
- Tilde Expansion
- Parameter Expansion
- Command Substitution
- Arithmetic Expansion
- Word Splitting
- Pathname Expansion

```
#!/usr/bin/env bash

foo=World
echo "Hello $foo" # Hello World
```

Expansion

: 유저가 입력한 커맨드에서 최종 명령문을 만들기 전에 이루어지는 일종의 가공 과정. 종류가 무려 7개

- Brace Expansion
- Tilde Expansion
- Parameter Expansion
- Command Substitution
- Arithmetic Expansion
- Word Splitting
- Pathname Expansion

`$((expression))`

```
#!/usr/bin/env bash

a=$((3 + 5))
echo $a
```

Expansion

: 유저가 입력한 커맨드에서 최종 명령문을 만들기 전에 이루어지는 일종의 가공 과정. 종류가 무려 7개

- Brace Expansion
- Tilde Expansion
- Parameter Expansion
- Command Substitution
- Arithmetic Expansion
- Word Splitting
- Pathname Expansion

globbing

```
#!/usr/bin/env bash

echo * # arithmetic case export foo for for greet if
 param pathname test
```


paco

while & until

```
while [ condition ]
do
  commands
done
```

참인 동안!

```
until [ condition ]
do
  commands
done
```

참일 때까지!

paco

실습

이전 실습을 수정하여 파일명 인자를 여러 개 받아 각각 파일에 대하여 이전 실습에서 하는 작업을 수행할 수 있도록 스크립트를 작성해보자

```
❯ ./query-file-loop /dev/stdout function . hello
/dev/stdout is some other file.
function is a regular file.
. is a directory.
hello does not exist.
```

Functions 함수

Functions

```
function_name()
{
  commands
  return // optional
}
```

```
function function_name()
{
  commands
  return // optional
}
```


paco

Functions

```
function_name()
{
  commands
  return // optional
}
```

```
function function_name()
{
  commands
  return // optional
}
```

argument는 일반 스크립트에서 다루듯이 하면 된다

Functions

```
function_name()
{
  commands
  return // optional
}
```

```
function function_name()
{
  commands
  return // optional
}
```

argument는 일반 스크립트에서 다루듯이 하면 된다

호출 또한 일반 스크립트 부르듯이 하면 된다

Functions

재귀함수도 가능

```
#!/usr/bin/env bash

printToN()
{
    if [ $1 -eq 0 ]; then
        return
    fi
    printToN $(($1 - 1))
    echo $1;
}

printToN $1
```

(샘플 코드 print-to-n 참고)

```
❯ ./print-to-n 10
1
2
3
4
5
6
7
8
9
10
```


paco

실습

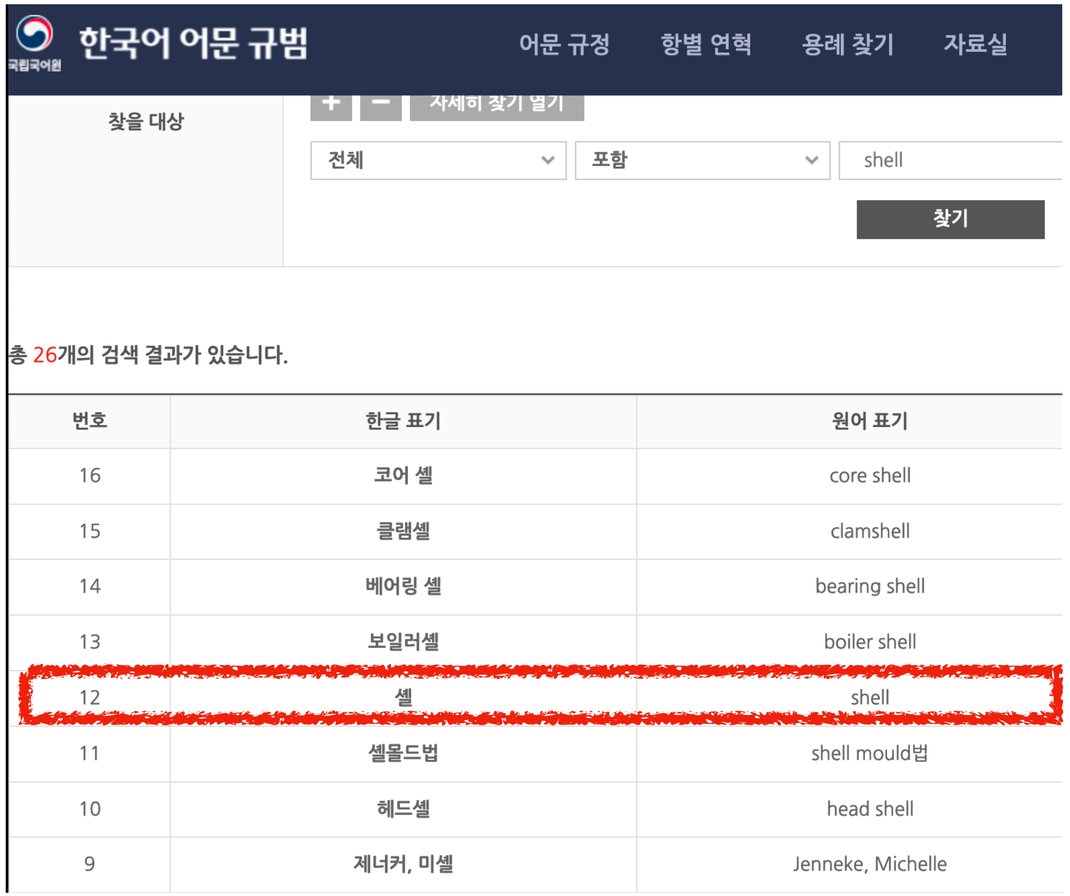
팩토리얼 계산하는 스크립트 짜기

```
❯ ./factorial 3
6
❯ ./factorial 5
120
❯ ./factorial 0
1
❯ ./factorial -1
Argument must not be negative
```

기타

셸 vs 쉘

셸 vs 쉘



https://kornorms.korean.go.kr/example/exampleList.do?regltn_code=0003

paco

man -> builtin(1)이 뜰 때

```
BUILTIN(1)                BSD General Commands Manual                BUILTIN(1)

NAME
     builtin, !, %, ., :, @, {, }, alias, alloc, bg, bind, bindkey, break, breaksw, builtins, case, cd, chdir,
     command, complete, continue, default, dirs, do, done, echo, echotc, elif, else, end, endif, endsw, esac,
     eval, exec, exit, export, false, fc, fg, filetest, fi, for, foreach, getopts, glob, goto, hash, hashstat,
     history, hup, if, jobid, jobs, kill, limit, local, log, login, logout, ls-F, nice, nohup, notify, onintr,
     popd, printenv, pushd, pwd, read, readonly, rehash, repeat, return, sched, set, setenv, settc, setty,
     setvar, shift, source, stop, suspend, switch, telltc, test, then, time, times, trap, true, type, ulimit,
     umask, unalias, uncomplete, unhash, unlimit, unset, unsetenv, until, wait, where, which, while -- shell
     built-in commands
```

$ man bash

에서 보자

paco

설정파일

- /etc/profile
- ~/.bash_profile
- ~/.bash_login
- ~/.profile
- ~/.bashrc

Login shell인지, interactive한지에 따라 source하는 설정파일이 다름.

man bash에서 “INVOCATION” 검색

여러 가지 셸들

sh (1979) – 배포. 유닉스 운영체제들의 셸로 사용

bash (1989) – sh 기반 GNU 프로젝트의 일환으로 작성된 free software

zsh (1990) – sh 기반 기존 sh에서 다수의 기능이 추가된 확장 셸

fish (2005) – Friendly Interactive Shell. 유저 친화적이고 다양한 기능들을 제공함

$(…) vs (…) vs $((…)) vs ((…)) vs ${…} vs {…}

- `$(...)` means execute the command in the parens in a subshell and return its stdout. Example:

```
$ echo "The current date is $(date)"
The current date is Mon Jul  6 14:27:59 PDT 2015
```

- `(...)` means run the commands listed in the parens in a subshell. Example:

```
$ a=1; (a=2; echo "inside: a=$a"); echo "outside: a=$a"
inside: a=2
outside: a=1
```

- `$((...))` means perform arithmetic and return the result of the calculation. Example:

```
$ a=$((2+3)); echo "a=$a"
a=5
```

- `((...))` means perform arithmetic, possibly changing the values of shell variables, but don't return its result. Example:

```
$ ((a=2+3)); echo "a=$a"
a=5
```

- `${...}` means return the value of the shell variable named in the braces. Example:

```
$ echo ${SHELL}
/bin/bash
```

- `{...}` means execute the commands in the braces as a group. Example:

```
$ false || { echo "We failed"; exit 1; }
We failed
```

https://stackoverflow.com/questions/31255699/double-parenthesis-with-and-without-dollar

paco

참고자료:

- 18년도 휠 세미나 Shell Script 부분
- 및 다수의 온라인 자료

paco