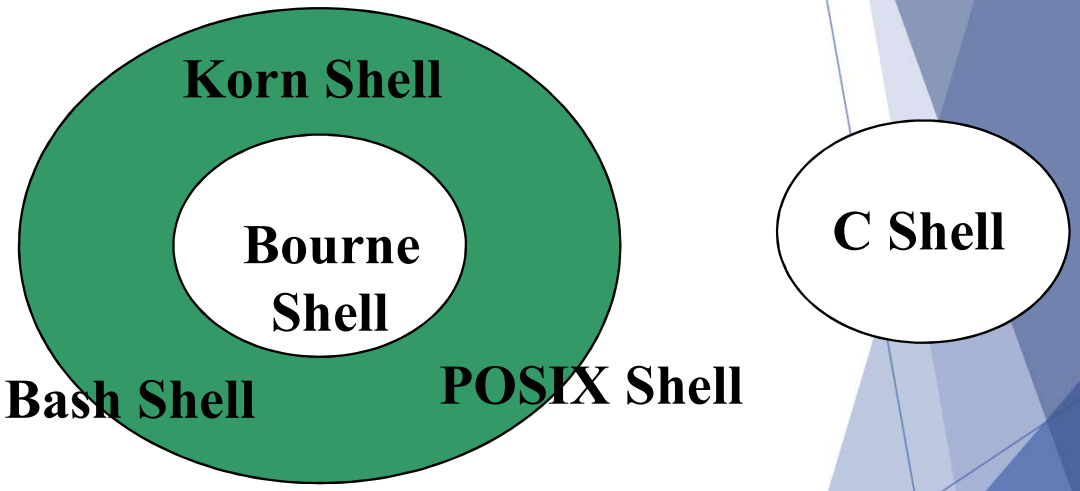
Linux System Administration

Shell and Shell Script

Shell과 Bash

- Shell
  - 운영체제와 유저 프로그램 사이에 존재
  - Bash 쉘이 가장 많이 사용
- Bash
  - 도구 프로그램 (Utility)
  - 명령어 해석기 (Command Interpreter)
    - 사용자 입력 명령어의 실행
  - 프로그래밍 언어
    - Shell Script에 있는 명령을 실행
- Shell Script
  - Shell Command 들을 일괄 처리할 수 있도록 절차와 순서에 따라 구성한 파일
  - 일반 프로그래밍 언어와 비슷하게 변수, 반복문, 제어문 등을 사용할 수 있음
  - 컴파일하지 않고 바로 실행할 수 있어 즉시 결과를 확인할 수 있다



Bash Shell Tips

- # : 주석
  - 이 표기 오른쪽은 실행되지 않음
- ; (세미콜론)
  - 여러 명령어를 한 줄에서 실행하고자 할 때 사용
  - 예) $ cmd1 ; cmd2 ; cmd3
- \
  - 명령어가 한 줄을 넘어 다음 줄에 연속될 때 사용
- echo {메시지}
  - {메시지}를 출력함
  - 개행(new line)을 막고자 하면 -n 옵션을 사용

Shell Script의 작성

- hello.sh 라는 이름으로 아래 코드를 입력
  - 쉘 스크립트 파일의 확장자는 아무 것이나 사용해도 상관 없으나 .sh를 부여하면 파일 명만으로도 쉘 스크립트 파일임을 인지할 수 있으므로 가능한 .sh 확장자를 부여

```
#!/bin/bash
date
echo Hello, Linux!
exit 0
```

- 1행: 특별한 형태의 주석으로 실행시 bash를 사용하겠다는 의미
- 2행: date 명령을 수행하고, 그 결과를 출력
- 3행: 메시지의 출력(echo) Hello, Linux! 라는 메시지를 출력함
- 4행: 종료 코드를 반환. 없어도 스크립트는 종료되지만 성공인지 실패인지 검증하기 위해서 반환해 주는 것이 좋음. 0은 성공값, 그 이외 값은 오류 혹은 예외

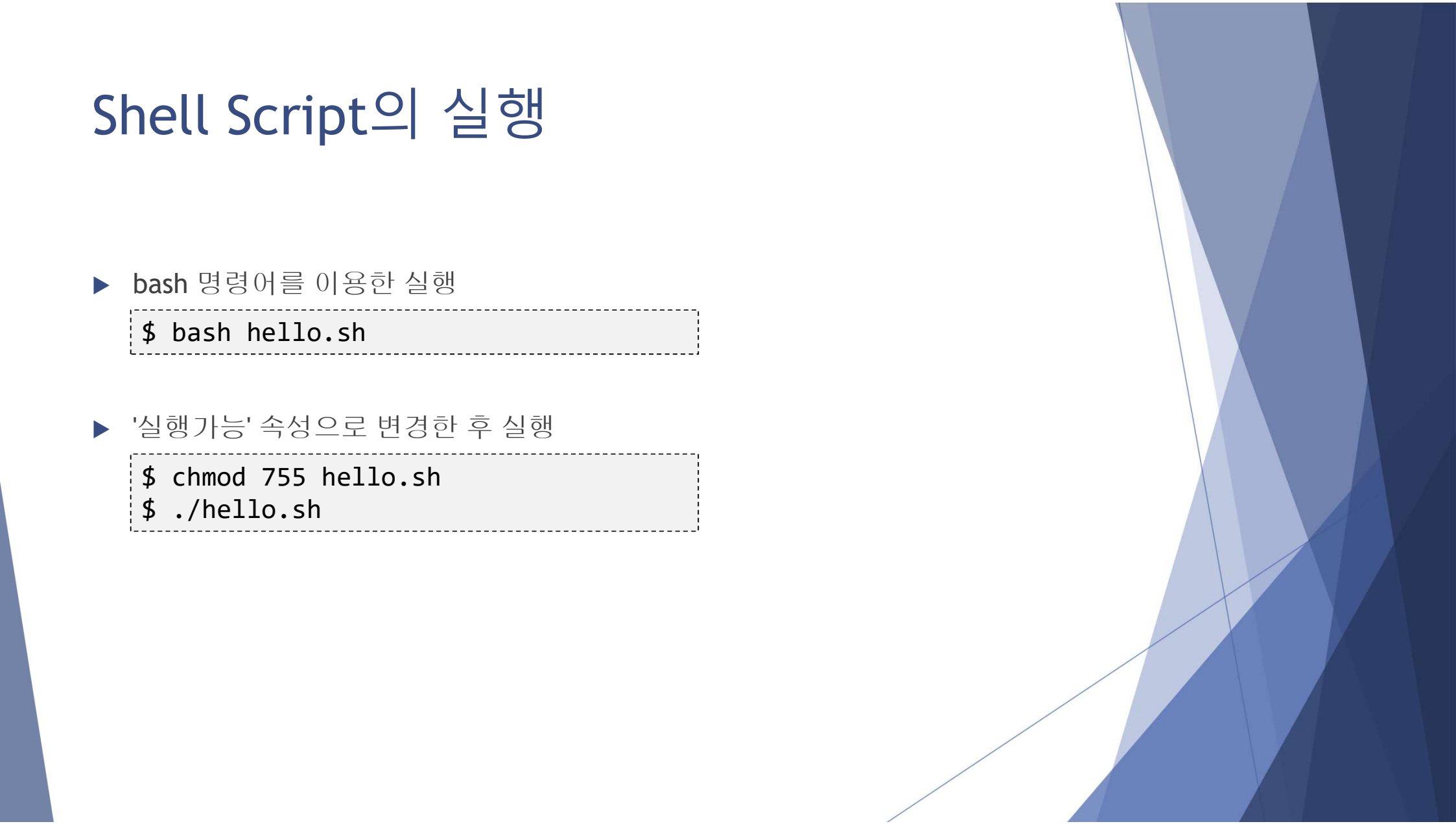
Shell Script의 실행

- bash 명령어를 이용한 실행

```
$ bash hello.sh
```

- '실행가능' 속성으로 변경한 후 실행

```
$ chmod 755 hello.sh
$ ./hello.sh
```



변수(Variables)

- 변수의 종류
  - User-Created Variables : 사용자 정의 변수
  - Keyword Shell Variables : 환경 변수
  - Read-Only Shell Variables
- 변수의 기본
  - 미리 선언할 필요는 없으며 처음 변수에 값이 할당되는 시점에서 자동으로 생성
  - 변수에 넣는 모든 값은 문자열로 취급
  - 변수명은 대소문자를 구분(Case Sensitive)
  - 변수 대입시 '=' 좌우에 공백이 있어서는 안됨
  - 대입할 값에 공백이 있을 경우에는 "로 묶어야 함
  - 변수 값을 출력하려면 앞에 $를 붙여야 함. 예) $myval
  - $ 문자를 출력하고자 한다면 앞에 \를 붙인다.

변수(Variables)

- 잘못된 변수 할당의 예

```
$ myval = Hello        # 오류: '=' 앞 뒤에는 공백이 없어야 함
$ myval=Hello, Linux   # 오류: 공백이 들어간 값은 ""로 묶어야 함
$ myval=7+5            # 할당은 정상적으로 되지만 '7+5'라는 문자열로 인식
```

- 정상적인 변수 할당의 예

```
$ myval=Hello
$ myval="Hello, Linux"
```

- 변수의 삭제

```
$ myval=
$ unset myval
```

변수(Variables)
: 수치 값의 계산 - expr

- 기본적으로 변수에 대입되는 값은 문자열
- 수치 계산을 위해서는 expr 키워드를 이용하며 전체 계산식은 역따옴표(`)로 묶어야 함
- 괄호, 곱셈표시(*) 앞에는 그 앞에 역슬래시(\)를 붙여줘야 함
- 연산자와 값 사이에는 공백을 둔다

```
num1=100
num2=$num1+200
num3=`expr $num1 + 200`
num4=`expr \( $num1 + $num3 \) \* 2`
```

사용자 입력
: read 명령어

- 사용자의 키보드 입력 값을 받음

```
# hello_name.sh
echo -n "Enter Your Name: "
read name
echo "Hello, $name!"
```

```
$ bash hello_name.sh
Enter Your Name: Sean
Hello, Sean!
```

명령어의 대치
: `, $()

- 역따옴표(`)는 리눅스 쉘 커맨드 명령어의 수행 결과를 대치한다
  - $(명령어)도 같은 역할을 수행한다

```
#!/bin/bash
# myhome.sh
echo Your Home Directory is `pwd`.
# echo Your Home Directory is $(pwd).
```

```
$ bash myhome.sh
Your Home Directory is /home/bituser.
```

환경변수

- 전체 환경 변수의 확인은 printenv 명령을 실행하면 출력(일부는 나타나지 않음)
- echo ${환경변수명}을 실행하면 해당 변수의 값을 확인할 수 있음
- 유용한 환경 변수
  - HOME : 사용자의 Home Directory
  - PWD : 현재 디렉터리
  - PATH : 실행 파일을 찾는 디렉터리 경로
  - USER : 현재 사용자의 이름
  - HOSTNAME : 호스트의 이름

```
#!/bin/bash
# names.sh
echo "User Name: $USER"
echo "User Directory: $HOME"
echo "Host Name: $HOSTNAME"
exit 0
```

파라미터 변수

- 실행하는 명령의 부분 하나하나를 변수로 지정. $0, $1, $2 등의 형태를 갖는다
  - $0은 스크립트 자신을 나타낸다

변수명	내용
$0	호출한 프로그램 이름
$1, $2, …, $9	개별 파라미터
$*	모든 파라미터
$#	파라미터의 개수
$$	수행중인 Shell의 PID
$?	지난번 프로세스의 exit 상태 (0이면 성공)

파라미터 변수

```
#!/bin/bash
# params.sh
echo Script Name: $0
echo Script PID: $$
echo Parameter Count: $#
echo 1st Parameter: $1
echo 2nd Parameter: $2
exit 0
```

```
$ bash params.sh Hello Linux
Script Name: params.sh
Script PID: 9314
Parameter Count: 2
Parameters: Hello Linux
1st Parameter: Hello
2nd Parameter: Linux
```

조건 분기
: if

```
# Syntax
if [ 조건1 ]
then
    조건1이 참일 경우의 실행
elif [ 조건12 ]
    조건2가 참일 경우의 실행
else
    조건을 만족하지 않을 경우의 실행
fi
```

```
#!/bin/bash
# ifelse.sh
if [ $1 = "Linux" ]
then
    echo Yes, I am
else
    echo No, I\'m not
fi
exit 0
```

```
$ bash ifelse.sh Linux
Yes, I am
$ bash ifelse.sh Windows
No, I'm not
```

조건 분기
: 비교 연산자

- 문자열 비교 연산자

비교 예	결과
"문자열1" = "문자열2"	두 문자열이 같으면 참
"문자열1" != "문자열2"	두 문자열이 같지 않으면 참
-n "문자열"	주어진 문자열이 NULL이 아니면 참
-z "문자열"	주어진 문자열이 NULL이면 참

- 산술 비교 연산자

비교 예	결과
수식1 -eq 수식2	두 수식이 같으면 참
수식1 -ne 수식2	두 수식이 다르면 참
수식1 -gt 수식2	수식1 > 수식2면 참
수식1 -ge 수식2	수식1 >= 수식2면 참
수식1 -lt 수식2	수식1 < 수식2면 참
수식1 -le 수식2	수식1 <= 수식2면 참
!수식	수식이 거짓이면 참

조건 분기
: 비교 연산자

```
#!/bin/bash
# compare.sh
if [ $# -ne 2 ]
then
    echo Need Two Numbers
elif [ $1 -eq $2 ]
then
    echo Equal
else
    echo Not Equal
fi
```

```
$ bash compare.sh
Need Two Numbers
$ bash compare.sh 100 200
Not Equal
$ bash compare.sh 100 100
Equal
```

조건 분기
: 파일 관련 조건표

조건	결과
-d 파일명	파일이 디렉터리면 참
-e 파일명	파일이 존재하면 참
-f 파일명	파일이 일반 파일이면 참
-r 파일명	파일이 읽기 가능하면 참
-s 파일명	파일 크기가 0이 아니면 참
-w 파일명	파일이 쓰기 가능 상태이면 참
-x 파일명	파일이 실행 가능 상태이면 참

조건 분기

: case ~ esac

- 참 혹은 거짓만 판별할 수 있는 if와는 달리 여러 가지 경우의 수를 판별하는 조건 분기문
- 경우의 수 표기는 '경우)' 로 표기
- 처리가 끝나면 세미콜론 두개(;;)를 써줘야 함

```
#!/bin/bash
# case1.sh
case $1 in
    baseball)
        echo 9 Players;;
    soccer)
        echo 11 Players;;
    basketball)
        echo 5 Players;;
    *)
        echo ?;;
esac
exit 0
```

관계 연산자
: AND(&&, -a), OR(||, -o)

- 조건문의 판별 조건이 복합적일 때 AND, OR 연산자를 이용한다
  - AND : && 혹은 -a 로 표기한다. 양쪽 조건이 모두 맞아야 참
  - OR : || 혹은 -o로 표기한다. 양쪽 조건 중 하나만 참이면 참

반복문
: for ~ in

- in 뒤에 입력된 값들을 차례로 받아와서 do 안에 있는 문장을 반복 실행한다

```
# Syntax
for 변수 in 값1, 값2, 값3 ...
do
    반복할 문장
done
```

- 기존 프로그래밍 언어와 비슷하게
  for ((i = 1 ; i <= 10 ; i++)) 형식으로 쓸 수도 있음
  이때 괄호가 두 개임을 유의

```
#!/bin/bash
# forin.sh
sum=0
for i in 1 2 3 4 5 6 7 8 9 10
do
    hap=`expr $hap + $i`
done
echo $hap
exit 0
```

반복문
: while

- 주어진 조건식이 참인 동안 계속 반복 실행

```
# Syntax
while [ 조건 ]
do
    반복할 문장
done
```

- until 문은 용도는 거의 같지만, 조건식이 거짓인 동안 (참이 될 때까지) 반복 실행
- 조건에 1을 주면 무한 반복

```
echo "Echo Back..."
read word
while [ $word != "enough" ]
do
    echo Echo: $word
    read word
done
echo Bye~
exit 0
```

반복문
: 반복문의 흐름 제어

- break
  - 반복문을 종료, 반복문 블록 이후의 문장 수행
- continue
  - 반복문의 조건으로 돌아감. 반복문의 다음 순번을 수행
- exit
  - 해당 프로그램 자체를 종료
- return
  - 함수 내에서 사용되며 함수를 호출한 곳으로 돌아가게 한다

사용자 정의 함수

- 사용자가 직접 함수를 작성하고 호출할 수 있음. 일련의 명령어를 저장
  - 함수의 정의

```
# Syntax
함수명() {
    수행할 명령들
}
```

```
#!/bin/bash
# myfunc.sh
sum() {
    echo `expr $1 + $2`
}
echo 10 더하기 20은?
sum 10 20 # sum 함수에 파라미터 10 20 전달
exit 0
```

  - 함수의 사용
    - 함수에 파라미터를 전달하려면 함수명 뒤에 차례로 붙여 전달한다
      이렇게 전달된 파라미터는 함수 내부에서 $1 $2 의 순서대로 사용된다