

삼성 청년 SW 아카데미

리눅스 셸 프로그래밍

<알림>

본 강의는 삼성 청년 SW아카데미의 콘텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

CLI Shell 종류

우리가 쓰고 있는 셸은 무엇일까요?

- 우분투 기본 CLI Shell : **Bash**
- 리눅스 사용자에게 가장 인기있는 CLI Shell

```
inho@inho-VirtualBox: ~  
inho@inho-VirtualBox:~$ ls  
a.c      gogo  test.tc  win  공개  문서  비디오  음악  
a.out   snap  ti.c    work 다운로드 바탕화면 사진  템플릿  
inho@inho-VirtualBox:~$
```

현재 사용중인 셸 확인

- /etc/passwd 파일에 기록되어 있음
- 사용자가 사용하는 CLI 셸 이름 확인 가능

```
inho@inho:~$ cat /etc/passwd | grep inho
inho:x:1000:1000:inho,,,:/home/inho:/bin/bash
inho@inho:~$
inho@inho:~$
```

CLI Shell 은 여러가지 존재

- `cat /etc/shells`

- 셸은 부팅하자마자,
어떤 셸로 실행될지 선택이 가능하다.
- **dash** : 데쉬, 데비안 암키스트 셸
 - 경량이다.
- **bash** : 배쉬 (정확하게는 본 어게인 셸)
 - 기능이 많다. 무겁다.

```
inho@inho-VirtualBox:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
inho@inho-VirtualBox:~$
```

bash

dash

어느게 파일 용량이 작을까?

```
inho@inho-VirtualBox:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
inho@inho-VirtualBox:~$
```

임베디드 리눅스

- dash 가 많이 사용 됨
- bash 도 따로 설치해서 사용 가능

PC 리눅스

- bash를 주로 사용함

bash 대신

dash 를 쓰면

셸 명령어가 달라지나?

- **대부분의 명령어는 똑같다.**
- dash를 쓰던, bash를 쓰던
일반적인 리눅스 사용에 있어 구분하기 쉽지않다.

Shell Script 개요

셸에서 실행하는 스크립트 프로그래밍 언어

- if, for, 변수 등을 사용하여 프로그래밍 가능

직접 살펴보자

- vi ~/.bashrc
- if 문도 보이고, else if 문도 확인할 수 있다.

```
inho@inho: /usr/bin
inho@inho: /usr/bin 119x32
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "${[ $? = 0 ]} && echo terminal || echo error" --title=Terminal -u $Urgency -t $Text'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi
```

파이썬 소스코드를 작성하였다.

- 파이썬 스크립트를 작성했다 라고 표현한다.

그럼 파이썬 스크립트의 실행기(Runtime)는?

JavaScript 소스코드를 작성하였다.

- JavaScript 스크립트를 작성했다 라고 표현한다.

그럼 JavaScript 스크립트의 실행기(Runtime)는?

bash shell script를 작성하였다.

그럼 bash shell script 의 실행기는??

```
#!/bin/bash

for ((var=0 ; var < 5 ; var++));
do
    echo $var
done
```

bash shell script 예시 (for문)

bash shell script를 작성하였다.

- bash shell script는 확장자가 sh 이다.
- 파일명 : test.sh 파일

임베디드 리눅스에 설치된
dash shell 에서 실행 가능할까?

→ 안된다.

이번 챕터의 목적!

1. bash shell script 를 작성 (프로그래밍)
2. bash shell 에서 동작시켜봄

→ 타인이 짤

bash script 소스코드를 이해할 수 있는 것

셸 스크립트는 자동화 프로그램 만들 때 쓴다.

- 예시 1) 자동 Script 만들기

- 파일명 예시 : backup.sh

- 실행하자마자 폴더가 자동 생성되고, 기존 파일이 백업되고, 백업된 파일이 서버로 전송하는 스크립트



셸 스크립트는 자동화 프로그램 만들 때 쓴다.

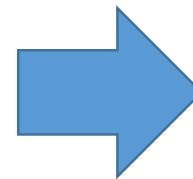
- 예시 2) 자동 세팅 Script 만들기

- 파일명 예시 : setup.sh

- 매번 초기 세팅해야 하는 반복해야 하는 작업을 shell script로 자동화 시킴

이 프로그램을 초기화 하시려면 동작시키려면

1. 기존 data 파일을 지우시고
2. init 이라는 명령어를 수행하시고
3. /etc/test 파일 설정에 BBQ=1 이라고 세팅하셔야합니다.



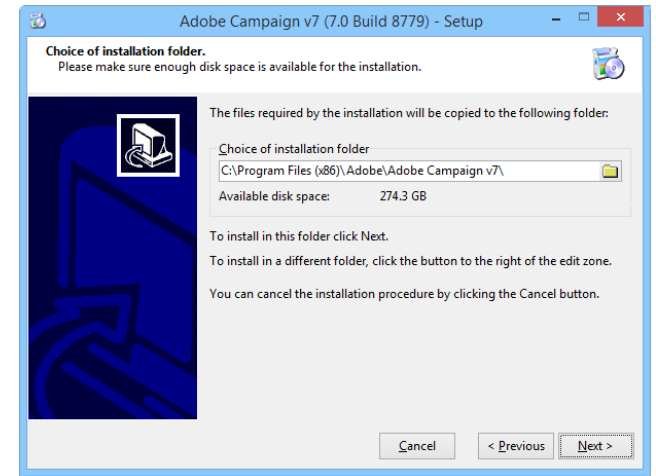
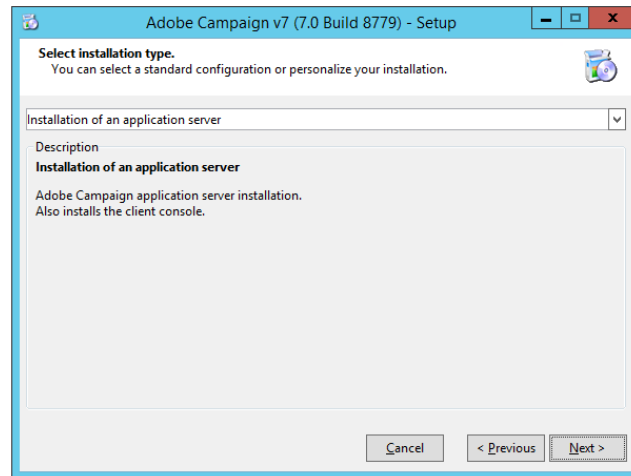
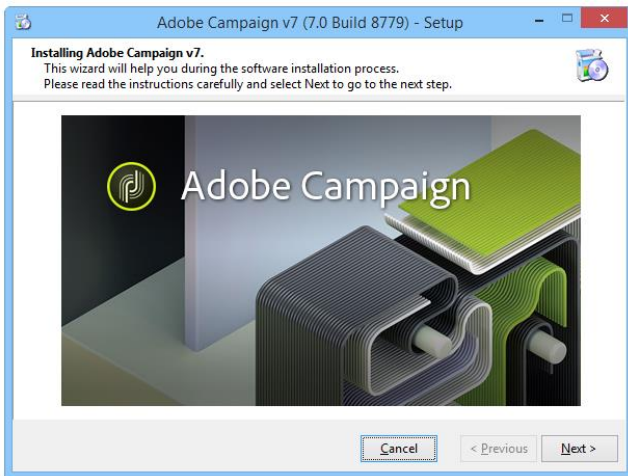
setup.sh
스크립트 파일 생성

셸 스크립트는 자동화 프로그램 만들 때 쓴다.

- 예시 3) 자동 설치 Script

- 파일명 예시 : install.sh

- 게임을 하나 제작하였음, 이것만 실행하면 PC 내 설치되게끔 Shell Script를 하나 만듦



bash shell script 대신 Python Script 사용

- Python 사용의 장점

- 코딩이 편하다.

- Python 단점

- python 실행기가 있어야 한다.
- 임베디드 환경에서 python 따로 준비해줘야한다.
(그런데, 그냥 준비하면 된다.)

```
#!/bin/bash
VAR1=10
VAR2=20
VAR3=30

if [ $VAR1 -lt $VAR2 -o $VAR2 -gt $VAR3 ]
then
    echo True
else
    echo False
fi
```

Bash Shell Script

```
money = 2000
card = True
if money >= 3000 or card:
    print("택시를 타고 가라")
else:
    print("걸어가라")
```

Python Shell Script

자동화 스크립트 제작은..

- bash 안해도 된다.
- python 만으로도 다 할 수 있다.

그런데 우리가 bash shell script 를 배우는 이유

- bash 로 짜여진 자동스크립트가 많아서이다.
- 10 년전만해도, 임베디드에서 Python은 거의 안쓰였지만
이제는 흔히 쓰인다.

임베디드는 다른 분야 대비
최신 트렌드에 민감하지 않아서,
고전스타일을 유지하는 경우가 많다.

Shell Script 체험

두 가지 규칙을 지켜주며 스크립트를 만들어보자

1. 모든 셸 스크립트 확장자 : `.sh`
2. (권장사항) 파일 맨 위에는 `#!/bin/bash` 를 적어줌
 - 이 문서는 bash 셸스크립트임을 알린다.
 - `#!/bin/bash` : bash 셸
 - `#!/bin/sh` : dash 셸
 - 쉬뱅 이라고 한다.

go.sh 쉘 스크립트를 제작하자

- shebang 추가하기
 - `#!/bin/bash`
- HI 메시지 띄우기
 - `echo "HI"`
- a.txt, b.txt, c.txt 파일 생성
- `ls -al ./*.txt` 수행
- `rm -r ./*.txt` 수행
- BYE 메시지 띄우기

```
inho@inho-VirtualBox:~/work2$ . go.sh
HI
-rw-rw-r-- 1 inho inho 0  3월 22 19:40 ./a.txt
-rw-rw-r-- 1 inho inho 0  3월 22 19:40 ./b.txt
-rw-rw-r-- 1 inho inho 0  3월 22 19:40 ./c.txt
BYE
inho@inho-VirtualBox:~/work2$
inho@inho-VirtualBox:~/work2$
```

실행방법 : “`source go.sh`”

방법 1. `source go.sh`

- 현재 bash 에서 go 셸스크립트 수행하기
- 가장 많이 사용되는 방법

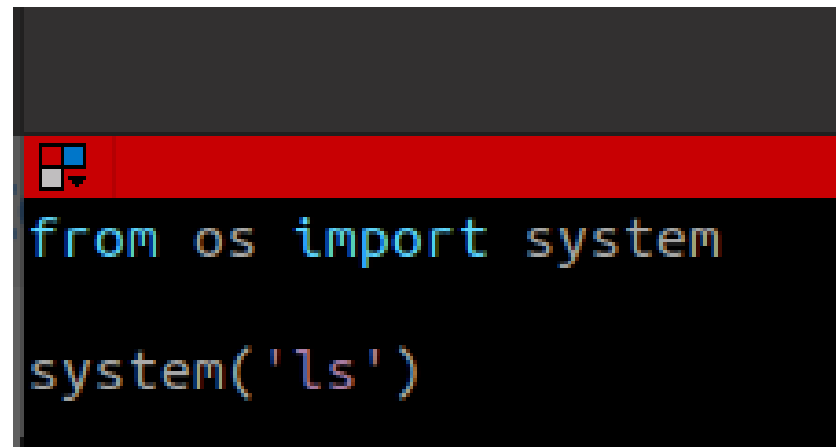
방법 2. `./go.sh`

- 실행권한만 주면 된다.
- 간헐적으로 사용되는 방법

go.py 스크립트를 제작하자

- HI 메세지 띄우기
- a.txt, b.txt, c.txt 생성
- ls -al ./*.txt 수행
- rm -r ./*.txt 수행
- BYE 메세지 띄우기

실행방법 : “python3 go.py”

A terminal window with a dark background and a red title bar. It contains two lines of Python code: 'from os import system' and 'system('ls')'.

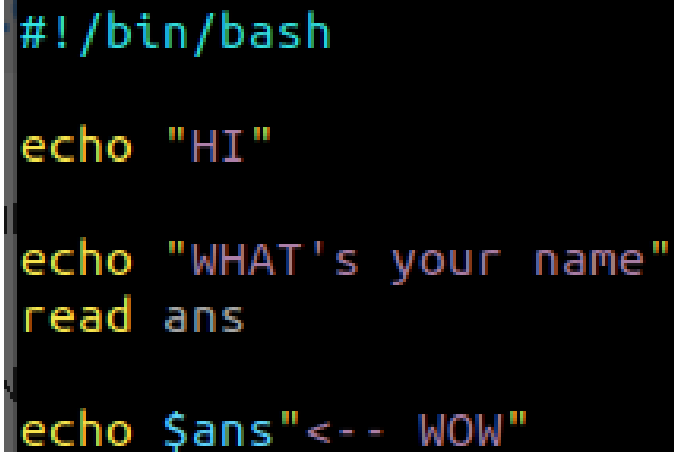
```
from os import system
system('ls')
```

파이썬 스크립트로
ls 명령어 수행 방법

변수

quest.sh 제작하기

- 간단한 질의 응답 스크립트 프로그램
- 출력 : echo
- 입력 : read



```
#!/bin/bash

echo "HI"

echo "WHAT's your name"
read ans

echo $ans" <- - WOW"
```

변수 만들기

- 변수이름=값
- 모든 값들은 문자열로 취급한다. (수로 취급 안함)

```
#!/bin/bash

bts=123
kfc=546

echo $bts + $kfc
```

```
#!/bin/bash

bts=123
kfc=546

hot=$kfc
god=kfc

echo $hot
echo $god
```

다음 출력 결과를 예상해보자.

- bts 1 ~ 5 까지 출력결과는?
- 이 중 한 줄의 명령어는 에러가 발생한다.

```
#!/bin/sh

bts1=100
bts2="100"
bts3=HOHO
bts4="HOHO"
bts5 = HAHA

echo $bts1
echo $bts2
echo $bts3
echo $bts4
echo $bts5
~
~
~
```

출력결과

- bts1 은 그대로 100 이 출력 됨
- bts2 는 그대로 100 이 출력 됨 (쌍 따옴표 생략)
- bts3 은 그대로 HOHO 출력 됨
- bts4 는 그대로 HOHO 출력됨
- bts5 는 에러 발생
 - 띄어쓰기가 존재하면 안된다.

```
#!/bin/sh

bts1=100
bts2="100"
bts3=HOHO
bts4="HOHO"
bts5 = HAHA

echo $bts1
echo $bts2
echo $bts3
echo $bts4
echo $bts5
~
~
~
```


첫 줄에는 100 이 출력된다.
다음 줄 부터 어떤 값이 출력될까?

```
#!/bin/sh

bts=100
echo $bts

bts=$bts+3
echo $bts

bts=bts+3
echo $bts

~
~
```

Shell Script 에서 변수 값을 문자열로 취급

- 출력결과 1 : 100
- 출력결과 2 : 100+3
- 출력결과 3 : bts+3

```
#!/bin/sh

bts=100
echo $bts

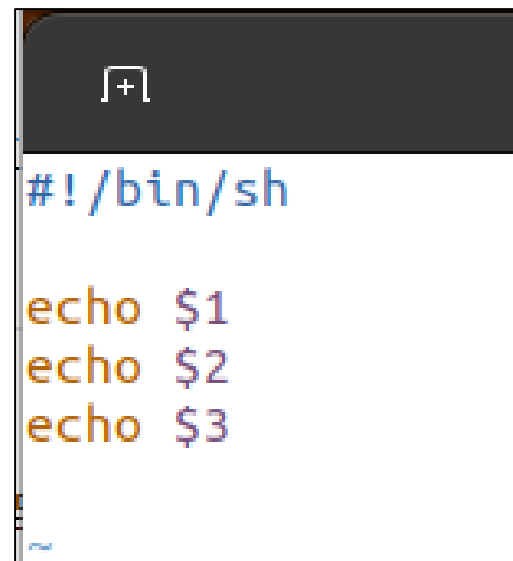
bts=$bts+3
echo $bts

bts=bts+3
echo $bts

~
```

Argument 변수

- `source quest.sh 100 200 abc` 입력시 출력결과
 - 100
 - 200
 - abc

A terminal window with a dark header bar containing a window icon and a '+' symbol. The terminal content shows a shell script being executed. The first line is a shebang `#!/bin/sh` in blue. The next three lines are `echo $1`, `echo $2`, and `echo $3`, all in orange. A blue tilde `~` is visible at the bottom left of the terminal area.

```
#!/bin/sh  
  
echo $1  
echo $2  
echo $3  
  
~
```

ok.sh 스크립트 제작

- Argument 2개 입력 필수
 - 첫 번째 인자값 : 수행할 명령어
 - 두 번째 인자값 : 수행할 명령어의 옵션
- 예시
 - source ./ok.sh ls al
 - 결과 : ls-al 이 수행됨

```
inho@inho-VirtualBox:~/work2$ source quest.sh ls al
합계 12
drwxrwxr-x  2 inho inho 4096  3월  22  21:13 .
drwxr-xr-x 28 inho inho 4096  3월  22  21:13 ..
-rwxrw-r--  1 inho inho   19  3월  22  21:13 quest.sh
inho@inho-VirtualBox:~/work2$
```

```
inho@inho-VirtualBox:~/work2$ source quest.sh uname r
5.8.0-45-generic
```

`$(())` 를 붙이면, 이 안에서 산술연산으로 처리됨

- 테스트를 해보자.

```
#!/bin/bash

bts=123

abc=$(( $bts + 123 ))

echo $abc
```

다음 스크립트를 제작하자

- **sum.sh 제작하기**

- Arg 1 : 숫자 1
- Arg 2 : 숫자 2
- Arg 3 : 숫자 3

- **출력결과 :**

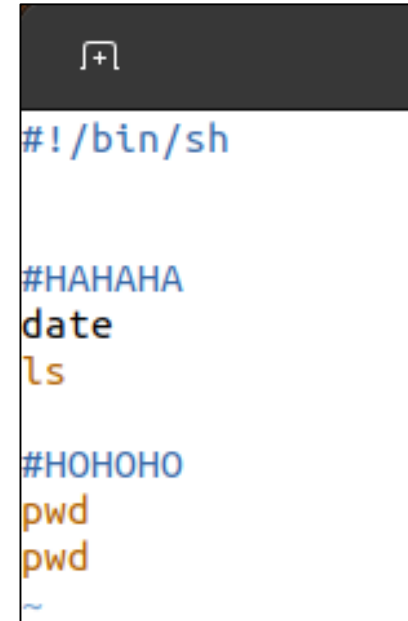
- SUM VALUE IS [] HAHA

- **예시**

- source ./sum.sh 100 200 50
- 결과 : SUM VALUE IS 250 HAHA

한줄 주석

- # 을 입력 후, 하고싶은 말 입력



```
#!/bin/sh

#HAHAHA
date
ls

#HOHOHO
pwd
pwd
~
```

sem.sh 제작하기

- /bin/bash 쉬뱅 추가하기
- arg1, arg2 에 숫자 입력
- 두 수의 합, 곱, 차 를 출력함
- 예시
 - source sem.sh 100 200
 - 출력결과
SUM : 300
GOP : 20000
CHA : -100

```
inho@inho-VirtualBox:~/work2$ source sem.sh 100 200
SUM:300
GOP:20000
CHA: -100
```


실행결과를 변수에 저장하기

- 변수=\$(셸 명령어)

```
#!/bin/bash

DATE=$(date)

echo $DATE "GOOD"
echo $DATE "HAHA"
```

```
inho@inho:~/work$ source ./quest.sh
Thu 03 Mar 2022 05:04:09 PM KST GOOD
Thu 03 Mar 2022 05:04:09 PM KST HAHA
inho@inho:~/work$
inho@inho:~/work$
```

도전

- 다음과 같이 출력하는 쉘 스크립트 작성

- meminfo 1 : MemTotal: xxxxx KB
- meminfo 2 : MemFree: xxxxx KB

```
inho@inho:~/work$ cat /proc/meminfo | grep MemFree
MemFree:          1322744 kB
inho@inho:~/work$ cat /proc/meminfo | grep MemTotal
MemTotal:         4019700 kB
inho@inho:~/work$
```

힌트, 쉘 명령어

if문

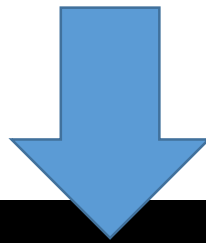
if문

- 띄어쓰기 조심해야한다.

```
inho@
inho@
#!/bin/bash
a=BTS
if [ $a = "BTS" ] ;then
    echo "BTS GOGO"
else
    echo "NO!!"
fi
```

잘 동작한다.

```
#!/bin/bash
a=BTS
if [ $a="BTS" ] ;then
    echo "BTS GOGO"
else
    echo "NO!!"
fi
```



```
if [ $a = "BTS" ] ;then
    echo "BTS GOGO"
else
    echo "NO!!"
fi
```

띄어쓰기로 에러

-lt

- less than

→ a가 50 보다 작으면

```
#!/bin/bash

a=5

if [ $a -lt 50 ] ;then
    echo "SMALL"
fi

~
```

그 밖에 수 비교

- **-eq**
 - “=” ← 이것은 문자열 비교이다. 수 비교가 아니다.
- **-gt**
 - greater then
- **-ne**
 - not equal
- **-ge**
 - 같거나 크다
- **-le**
 - 작거나 같다.

소스코드를 이해해보자.

- or : ||
- and : &&

```
inho@inho: ~/work 59x22
#!/bin/bash

a=5

if [ $a -gt 50 ] || [ $a -eq 5 ] ;then
    echo "LUCKY"
fi

~
```

수 하나 입력

수가 10보다 크고 30보다 작으면 “good” 출력

그렇지 않고 (elif)

수가 40보다 크고 50보다 작으면 “omg” 출력

전부 아니면 (else)

“wow” 출력

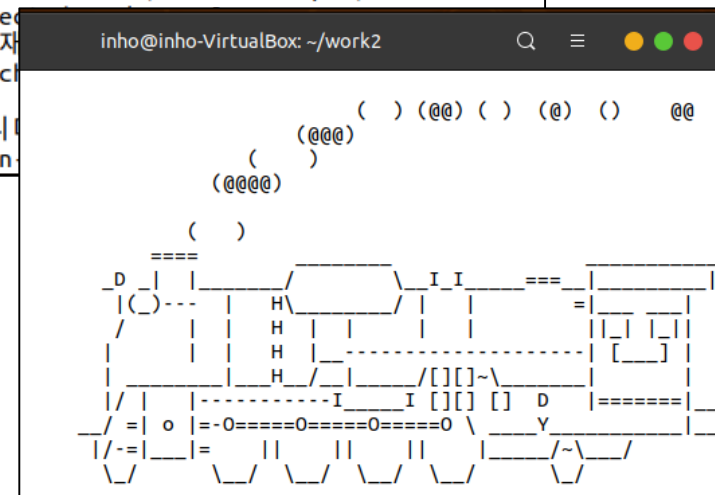
ID 와 PASSWORD 를 정확하게 입력해야 동작하는 프로그램

- **secret.sh 파일 생성**
 - Arg1 : ID 문자열 입력
 - Arg2 : PASS 문자열 입력
- **ID 가 “KFC” 이고, 비밀번호가 “1234” 인 경우만 다음 명령어를 수행한다.**
 - kfc.txt 파일을 생성
 - kfc 내용에 “i love my eyes” 내용 입력
 - 화면 clear
 - cat 으로 kfc 파일 세 번 반복 출력
 - “sleep 1 “ 명령어 수행 (1초 대기)
 - 화면 clear
 - echo 명령어로 “keep secret, BYE” 출력
 - kfc.txt 파일 삭제

run.sh 쉘 스크립트 제작

- argument 에 Type을 입력 받음
- Type : “dd”
 - date 명령어 수행
- Type : “sl”
 - sl 프로그램 설치 후 실행 (apt install -y && run)
 - sl 프로그램 삭제 (apt purge -y)

```
inho@inho-VirtualBox:~/work2$ source run.sh sl
패키지 목록을 읽는 중입니다... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다... 완료
다음 새 패키지를 설치할 것입니다:
sl
0개 업그레이드, 1개 새로 설치, 0개 제거 및 8개 업그레이드 안 함.
12.7 k바이트 아카이브를 받아야 합니다.
이 작업 후 60.4 k바이트의 디스크 공간을 더 사용하게 됩니다.
받기:1 http://mirror.kakao.com/ubuntu focal/universe amd64 sl amd64
[kB]
내려받기 12.7 k바이트, 소요시간 0초 (336 k바이트/초)
Selecting previously unselected package sl.
(데이터베이스 읽는중 ...현재
Preparing to unpack .../arc
Unpacking sl (5.02-1) ...
sl (5.02-1) 설정하는 중입니다
Processing triggers for man
```



for문, 함수, 배열

셸 스크립트 목적이 주로 자동프로그램 만들기 이기에 자주쓰는 문법은 다음과 같다.

- echo 출력
- 실행 파라미터
- if문

자주쓰지 않는 문법은 다음과 같다.

- printf, for문, 함수, 배열

→ 자주 쓰이지 않는 문법을, 체험 위주로 학습해보자.

C언어의 printf와 비슷

- echo 와 달리 개행문자를 넣어주어야 개행 처리가 된다.
- 예시
 - printf "HI HI \n"
 - printf "NAME : %s \n" "INHO"

```
info@info-VirtualBox: ~/wor
#!/bin/bash
a=100
b=200
c=300
printf "HI %d HOHO %d CCCC : %d \n" $a $b $c
```

배열 만들기

- 배열 만들때는 () 를 사용
- 배열값을 출력할 때는 {} 괄호 필수

```
#!/bin/bash

arr=(10 20 30 40)

echo ${arr[0]}
echo ${arr[1]}
echo ${arr[2]}
echo ${arr[3]}

~
```

HI 10회 출력하는 코드

```
#!/bin/bash

for ((i = 0; i<10; i++))
do
    echo "HI"
done

~
```

```
#!/bin/bash

for ((i = 0; i<10; i++));do
    echo "HI"
done

~
```

함수 만들기

- 함수 호출시, 함수 이름만 입력하면 된다.

```
#!/bin/bash

abc()
{
    printf "HIHI\n"
}

printf "START\n";
abc
abc
```


환경변수

환경변수 : 셸에 저장되는 변수

- 사용자, Process, 리눅스 자체 등
다 같이 사용하는 변수이다.

챕터의 목적

- 환경변수 사용법을 익힌다.
- 이 환경변수를 사용하는 스크립트를 제작해보자.

- 환경변수 전체 읽기

- `printenv` 명령어

- 하나만 읽기

- `echo $변수명`
 - `echo $SHELL`
 - 현재 사용중인 셸 경로를 저장한 변수
 - `echo $PWD`
 - 현재 디렉토리가 저장된 변수
 - `echo $HOME`
 - 홈 디렉토리 저장된 변수

값 저장하기

- `export [변수]=[값]`

KFC변수에 “HELLO” 값을 저장 후,
변수값을 출력해보자.

```
inho@inhopc:/home$ printenv | grep KFC
KFC=HELLO
inho@inhopc:/home$
```

터미널을 종료 했다가 다시 켜보자

- 원격접속터미널이라면 접속을 끊었다가 다시 켜다.

이제 다시 KFC 변수값을 읽어본다.

- 변수 값 유지가 안된다.

~/.bashrc

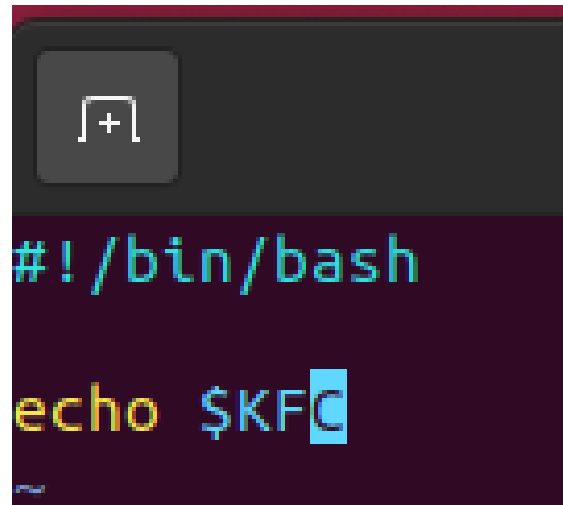
- bash 셸이 시작되자마자 자동으로 시작되는 bash 스크립트 파일

환경변수값을 지속적으로 유지하려면?

- ~/.bashrc 파일에 export를 추가한다.

1. KFC 변수 세팅 명령어를 .bashrc 파일에 등록해두자.
 - ~/.bashrc 파일 맨 마지막에 export 명령어를 추가한다
2. 터미널을 껐다 켜다.
3. 환경 변수 값이 유지되는지 확인한다.

아래와 같이 코딩하면,
환경 변수값을 읽을 수 있다.



```
#!/bin/bash  
echo $KFC
```


bash 쉘 스크립트를 작성한다.

- 만약 환경변수값 KFC 가 HELLO 값을 가진다면,
 - OH GOOD HI 를 출력한다.
- 그렇지않으면
 - OH MY GOD 을 출력한다.

실전, 웹 스크립트 분석하기

챕터 목적

- 실제 사용중인 쉘 스크립트 파일을 분석하고 이해해보자.

시작하기 전 상식 1

- alias는 별명을 만들어내는 명령어이다.

실습

- alias f1='mkdir ./bts;ls'
- alias f2='rm -rf ./bts;ls'

- f1 이라고 입력해보자.
- f2 이라고 입력해보자.

시작하기 전 상식 2

- ll을 입력 해보자.
 - ls -al 명령어를 쉽게 쓸 수 있다.

ll은 bashrc 파일에서 만든 alias 이다.

→ ~/.bashrc 스크립트 파일에서 ll 관련 alias를 검색해보자.

bashrc 파일에는
다음과 같이 alias 지정하는 스크립트가 있다.

```
# some more ls aliases  
alias ll='ls -alF'  
alias la='ls -A'  
alias l='ls -CF'
```

if 문 옵션

- -n “문자열” : 문자열 길이가 0 보다 클 때
- -z “문자열” : 문자열 길이가 0 일 때
- -x : 파일이 존재하고, 권한이 실행 (+x) 일 때
- -f : 파일이 존재하고, Regular 파일 일 때

```
if [ -n "$force_color_prompt" ]; then  
    if [ -x /usr/bin/tput ] && tput se
```

```
if [ -f ~/.bash_aliases ]; then  
    . ~/.bash_aliases  
fi
```

case 문

- c 언어의 switch case 문과 동일한 구조이다.

```
case 문자열 in
문자열1|문자열2)
    소스코드1
;;
문자열3)
    소스코드1
;;
esac
```

```
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac
```


변수 선언방법

- `[변수명]=[값]`
 - ex) KFC=119

변수 제거 방법

- `unset [변수명]`
 - ex) unset KFC

```
• unset color_prompt force_color_prompt
```

정확한 내용 분석이 아닌

문법적으로 어떤 구조로 되어있는지
각자, 문법 구조 분석을 해보자.

```
inho@inhop: ~/work
# (ISO/IEC-6429). (Lack of such support is extre
# a case would tend to support setf rather than
color_prompt=yes
else
color_prompt=
fi
fi
if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m
33[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h:
    ;;
*)
    ;;

```

도전, 백업 스크립트 만들기

3 시간에 한번 씩

Work 디렉토리를 백업하는 스크립트 만들기

1. `~/work` 디렉토리를 압축 후, 특정 디렉토리에 백업하는 스크립트를 만든다.
 - 파일명 : `backup.sh`
2. **crontab** 을 사용하여,
주기적으로 `backup.sh` 파일을 수행한다.

먼저 `crontab` 에 대해서 알아보자.

crontab (크론탭)

- 원하는 시간 / 조건에 특정 명령어를 수행 시키기 위해
만들어야 하는 파일

cron 데몬 (크론)

- crontab 문서에 적은 내용대로 수행해주는 데몬

crontab 테스트를 위한 스크립트 파일 하나 작성하기

- 파일명 : ~/test.sh
- 실행 권한 부여 : `chmod +x ~/test.sh`

date '+%H:%M:%S'

- 현재시간 출력 명령어

스크립트 내용

- 홈 디렉토리에 기존에 생성된 파일을 삭제
- 홈 디렉토리에 빈 파일을 하나 생성 (파일명은 test-시간)
- 목록을 출력한다.

```
FILENAME=$HOME/test-$(date '+%H:%M:%S')  
  
rm -rf $HOME/test-*  
touch $FILENAME  
ls  
~  
~
```

1 분에 한번씩 수행하기

- `sudo vi /etc/crontab`

```
info@inhopc: ~  
# that none of the other crontabs do.  
  
SHELL=/bin/sh  
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin  
  
# Example of job definition:  
# ----- minute (0 - 59)  
# | ----- hour (0 - 23)  
# | | ----- day of month (1 - 31)  
# | | | ----- month (1 - 12) OR jan,feb,mar,apr ...  
# | | | | ----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat  
# | | | | |  
# * * * * * user-name command to be executed  
17 * * * * root    cd / && run-parts --report /etc/cron.hourly  
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )  
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )  
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )  
#  
* * * * * inho    /home/inho/test.sh  
~  
~  
~
```

정상 동작 확인 : **service cron status**

- 에러메세지가 있다면 crontab 파일을 수정한 후, cron 데몬을 재시작한다.
- 재시작 명령어 : **service cron restart**

```
inho@inhopc:~$ service cron status
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2022-03-05 23:34:36 KST; 1s ago
     Docs: man:cron(8)
  Main PID: 1883 (cron)
    Tasks: 1 (limit: 9463)
   Memory: 360.0K
    CGroup: /system.slice/cron.service
            └─1883 /usr/sbin/cron -f

3월 05 23:34:36 inhopc systemd[1]: cron.service: Succeeded.
3월 05 23:34:36 inhopc systemd[1]: Stopped Regular background program processing daemon.
3월 05 23:34:36 inhopc systemd[1]: Started Regular background program processing daemon.
3월 05 23:34:36 inhopc cron[1883]: (CRON) INFO (pidfile fd = 3)
3월 05 23:34:36 inhopc cron[1883]: Error: bad command; while reading /etc/crontab
3월 05 23:34:36 inhopc cron[1883]: (*system*) ERROR (Syntax error, this crontab file will be ignored)
3월 05 23:34:36 inhopc cron[1883]: (CRON) INFO (Skipping @reboot jobs -- not system startup)
inho@inhopc:~$
```

해당 그림과 같이 에러 메세지가 나오는 경우, crontab 파일을 수정해야한다.

crontab 예시

* * * * * : 매 분마다 수행

* 7 * * * : 매일 7시마다 수행

*/3 * * * * : 매 3분 마다 수행

* */3 * * * : 매 3시간 마다 수행

```
# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed
```

1. backup.sh 제작하기

- ~/work의 모든 파일들을 압축한다.
 - 파일명 : 현재시간.tar.gz
- 압축된 파일을 /data/backup 디렉토리로 이동시킨다.
 - 만약 폴더가 없다면 폴더를 생성한다.
 - 만약 기존에 존재하는 파일이 있다면, 덮어쓴다.

2. 3 시간에 한번 씩, backup.sh 스크립트 자동 실행

- crontab 사용

내일 방송에서 만나요!

삼성 청년 SW 아카데미