

삼성 청년 SW 아카데미

리눅스 쉘 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.

프로그램 설치

챕터의 포인트

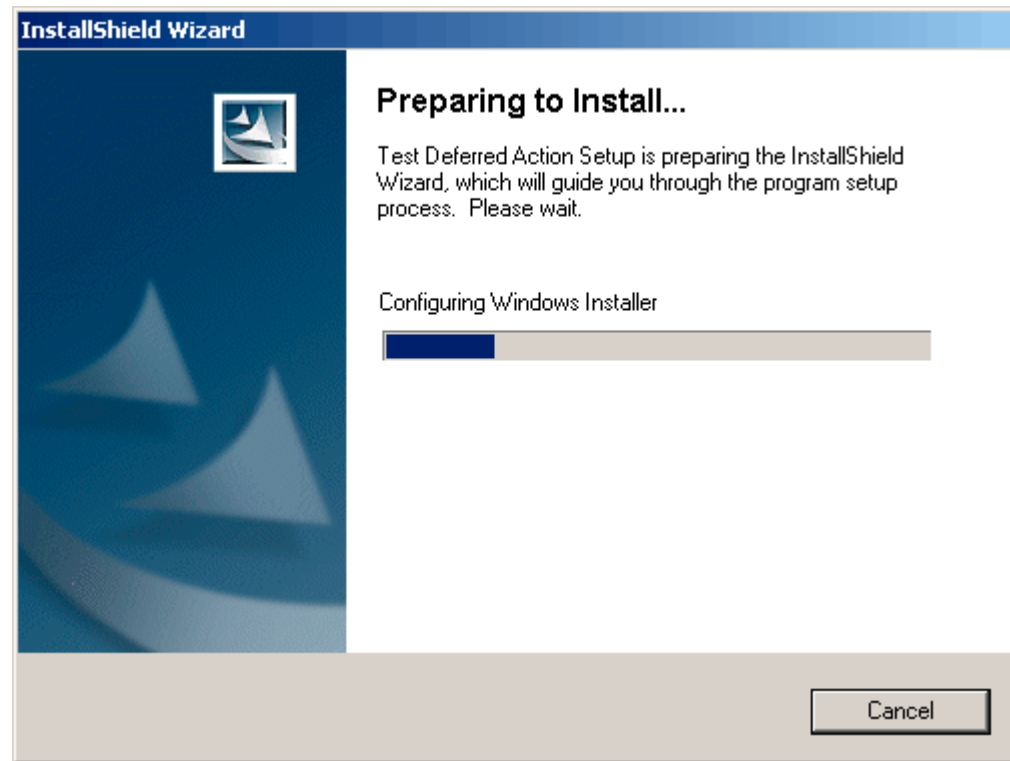
- 리눅스 배포 방법
- zip / tar
- node Binary 설치하기
- Source Code로 설치
- dpkg 설치

리눅스 배포 방법

App 설치

- 윈도우 프로그램 배포
 - 설치 프로그램 형태로 배포가 잦다.
- 리눅스 프로그램 배포
 - 압축 파일 형태로 배포하는 경우가 잦다.
 - 압축파일 다루는 방법은 윈도우보다 중요도가 높다.

리눅스를 다룰 줄 안다면,
프로그램은 쉽게 설치할 줄 알아야함



리눅스에서는 여러가지 배포 방법을 사용한다.

1. 패키지관리 도구로 배포하기

- apt 로 배포
- 우분투 소프트웨어 배포

2. Binary 파일을 tar.xz 로 압축하여 배포

- 다양한 방식으로 압축된 파일을 풀어서 설치

3. Source Code를 압축하여 배포

- 직접 GCC Build 하여 사용해야 한다.

4. dpkg 파일로 배포

다음과 같은 순서대로 학습 예정

1. 리눅스 압축 파일 다루는 방법을 연습한다.

- 학습 이유 : 리눅스에서는 압축 형태로 프로그램을 배포하는 경우가 잦기 때문

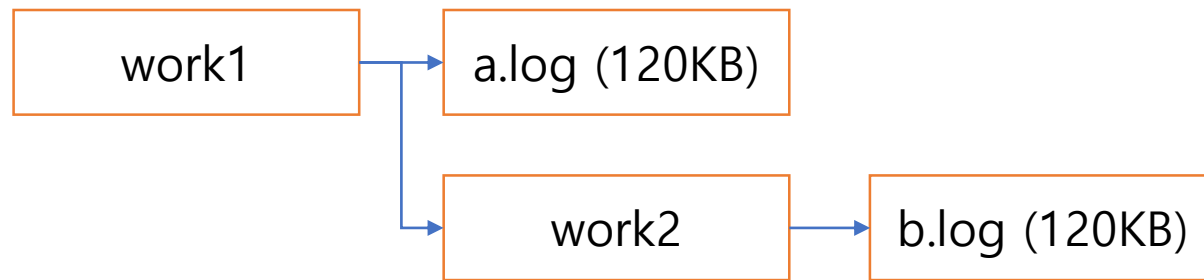
2. 여러가지 방법으로 배포된 프로그램들을 설치해본다.

- 패키지 관리도구에 대한 상세한 정보는 차후 학습 예정

zip / tar

아래와 같이 실습환경을 구성하자

- work1 , work2 디렉토리 생성
- work1, work2 각각에 120kB인 파일 생성
 - 1글자 = 1byte
 - 1024 * 120 의 글자 필요
 - 용량 확인
`$ls -alh ./a.log`



- 아래의 명령어를 입력한 후, 압축 파일의 용량을 확인하자
 - 1. `$zip all1.zip a.log`
 - -> all1.zip 용량 확인
 - 2. all1.zip 삭제 후, `$zip all2.zip ./*`
 - -> all2.zip 용량 확인 후, GUI Shell 로 all2.zip 열어서 내용 확인하기
 - 2. all2.zip 삭제 후, `$zip -r all3.zip ./*`
 - -> all3.zip 용량 확인 후, GUI Shell 로 all3.zip 열어서 내용 확인하기

unzip 명령어 사용

- **unzip [압축 파일명]**
 - 현재 디렉토리로 압축을 푼다.
 - \$unzip all3.zip -> a.log / b.log 가 나온다. unzip_a.log, unzip_b.log 로 각각 rename 하자
- **unzip [압축 파일명] -d [디렉토리]**
 - 특정 디렉토리로 압축을 푼다.
 - \$unzip all3.zip -d ./tt -> 현재 디렉토리에 ./tt 디렉토리 생성 후 그 안에 압축해제

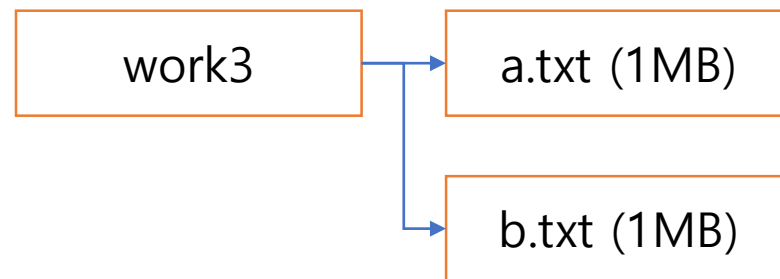
echo 명령어 사용법

- `$echo ABC > a.txt`
 - 출력결과를 a.txt 파일을 생성 후 적는다.
 - 기존 파일이 있다면, 내용을 모두 지우고 새로 교체
- `$echo ABC >> a.txt`
 - 출력결과를 a.txt 아랫줄에 덧붙인다.

셸스크립트 명령어 for 문 사용하기

- 오른쪽처럼 디렉토리와 파일을 준비하자.
- 1MB 파일 만들기

```
$for ((i=0; i<1024*1024/2; i++)) do echo A >> a.txt; done
```



리눅스 “단일 파일” 압축

- “원본 파일”이 “압축 파일”로 변경된다. (*zip 명령어는 원본 파일 그대로 압축 파일 생성)
- 단 하나의 파일만 압축 가능하다.

gzip

- 압축 : gzip [파일명]
- 압축해제 : gunzip [파일명]

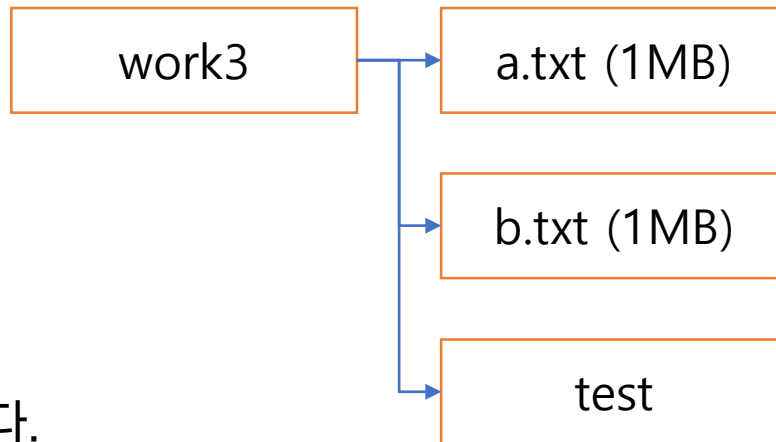
XZ

- 압축 : xz [파일명]
- 압축해제 : xz -d [파일명]

a.txt 는 gzip, b.txt 는 xz 로 압축해서
용량을 확인하자

하나의 파일로 만들기

- 압축 되지 않는다.
- **tar -cvf [파일명.tar] [파일들]**
 - -c : 하나로 모은다.
 - -f : 파일이름을 지정한다.
 - -v : 진행 상황을 보여준다.
 - 예시 : `tar -cvf ./test.tar ./*`
- 압축을 풀 디렉토리를 하나 만들자.
 - `$mkdir test`
 - tar 는 자동으로 디렉토리를 만들어 주는 기능이 없다.
- **tar -xvf [파일명.tar] -C [압축 풀 디렉토리 경로]**
 - -x : tar 해제
 - -C : 경로 지정



리눅스에서 많이 사용되는 압축 방법 (인기순위)

- tar + xz
- zip
- tar + gz
- tar + bz2

압축률 : zip < gz < bz2 < xz

- 압축률이 높을수록, 속도는 느려진다.

tar 사용법

- `tar -cvf` : 모으기
- `tar -xvf` : 모은것 풀기

tar.xz 사용법

- `tar -Jcvf` : 한 파일로 모으고, 압축하기
- `tar -Jxvf` : 압축풀고, 모은것 풀기

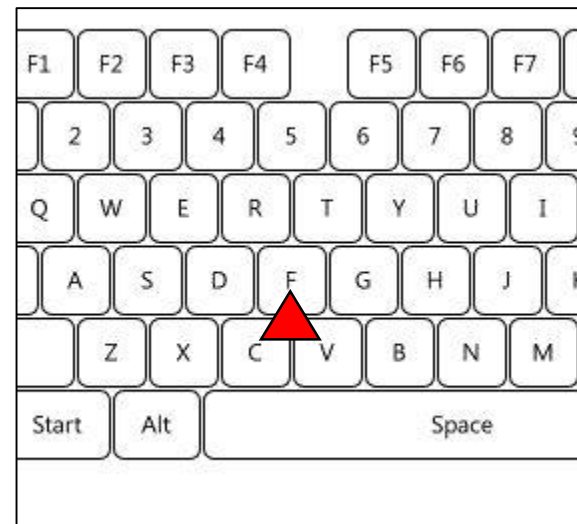
디렉토리에 푸는 방법

- 디렉토리 직접 만들고, `-C [경로]` 옵션 사용하기

`$tar -Jcvf ab_xz.tar a.txt b.txt` : tar.xz 로 압축

`$ls -alh` : 엄청난 압축률을 볼 수 있음

`$tar -Jxvf ab_xz.tar -C ./test` : ./test에 압축 해제



암기방법추천 : 키보드 삼각형

다음과 같은 과정을 진행하자

- 오른쪽과 같이 디렉토리와 파일을 준비하자

1. tar.xz로 압축하기

- 옵션 Jcvf

2. test1 에 압축파일 해제

- 옵션 Jxvf

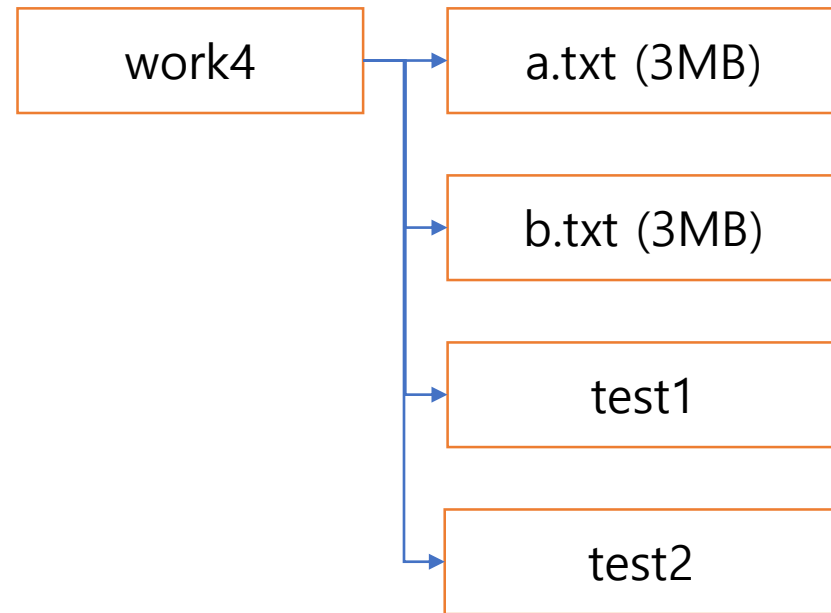
3. tar.gz 으로 압축하기

- 옵션 zcvf

4. test2에 압축파일 해제

- 옵션 zxvf

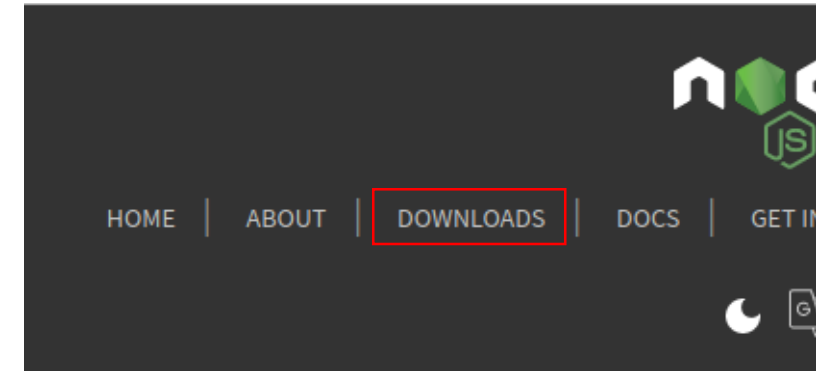
5. tar.xz 와 tar.gz의 압축률 비교하기



node Binary 설치하기


node.js 를 설치해보자.


- nodejs.org 접속해서 다운로드 페이지로 이동
- 오른쪽 버튼을 눌러서 URL 복사하기 (클릭하여 다운로드 하지 말것)




LTS
Recommended For Most Users

Current
Latest Features


Windows Installer
node-v14.16.0-x64.msi


macOS Installer
node-v14.16.0.pkg


Source Code
node-v14.16.0.tar.gz

Windows Installer (.msi)
Windows Binary (.zip)
macOS Installer (.pkg)
macOS Binary (.tar.gz)
Linux Binaries (x64)
Linux Binaries (ARM)
Source Code

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v14.16.0.tar.gz	

PC용 리눅스

임베디드 리눅스

wget 명령어로 다운로드 받기

- wget [URL]

```
inho@inho-VirtualBox:~/work$ wget https://nodejs.org/dist/v14.16.0/node-v14.16.0-
linux-x64.tar.xz
--2021-03-07 20:50:46-- https://nodejs.org/dist/v14.16.0/node-v14.16.0-linux-x6
4.tar.xz
nodejs.org (nodejs.org)을(를) 해석하는 중... 104.20.22.46, 104.20.23.46, 2606:47
00:10::6814:172e, ...
접속 nodejs.org (nodejs.org)|104.20.22.46|:443... 접속됨.
HTTP 요청을 전송했습니다. 응답을 기다리는 중입니다... 200 OK
길이: 21385948 (20M) [application/x-xz]
다음 위치에 저장: `node-v14.16.0-linux-x64.tar.xz'

node-v14.16.0-linux 100%[=====>] 20.39M 9.50MB/s / 2.1s

2021-03-07 20:50:48 (9.50 MB/s) - `node-v14.16.0-linux-x64.tar.xz' 저장됨 [21385
948/21385948]
```

tar.xz 압축 파일을 해제하자

- **/usr/local/lib/nodejs 에 디렉토리를 하나 생성하자**
 - `$sudo mkdir -p /usr/local/lib/nodejs` : -p 디렉토리 생성 시 필요한 경우 부모 디렉토리까지 생성
 - /usr/local 디렉토리 : 사용자 설치를 하는 디렉토리 (C:\WProgramFiles으로 비유될 수 있음)
 - 반드시 위 디렉토리에 하지 않아도 된다. 다만 권장하는 디렉토리 이다.
- **생성한 디렉토리에 압축을 해제한다. (옵션 -Jxvf)**
 - 직접 해보자.
 - nodejs 의 버전에 주의하자 (ls -al 로 압축파일명 확인 후 진행)

설치된 경로에 들어가자

- `cd [설치경로]/bin/`
- 명령어 실행해보기 : `./node -v`

```
inho@inho-VirtualBox: /usr/local/lib/nodejs/node-v14.16.0-linux-x64/bin$ ./node -v
v14.16.0
inho@inho-VirtualBox: /usr/local/lib/nodejs/node-v14.16.0-linux-x64/bin$
```

어떤 경로에서나 잘 실행이 되도록 심볼릭 링크 걸기

- `cd [설치경로]/bin/` 이 디렉토리에서 실행하기
 - `$sudo ln -s $PWD/node /usr/bin/node`
- HOME 디렉토리에서 `node -v` 를 수행해보자.

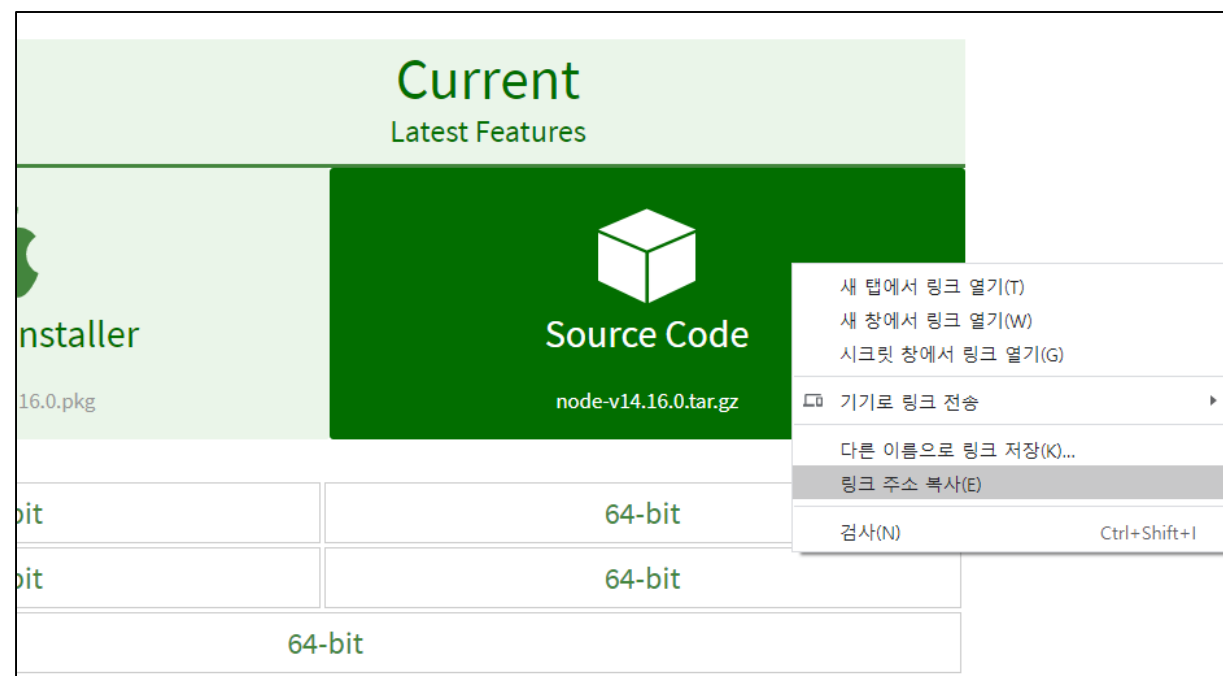
Source Code로 설치

기존에 설치했던 nodejs 제거

- 심볼릭 링크 제거 (/usr/bin/node)
- Binary 디렉토리 제거 (/usr/local/lib/nodejs)
- wget으로 다운로드 받은 파일 제거

nodejs.org 방문

- Source Code 다운로드 URL 복사하기
- wget 명령어로 다시 다운로드 받기 (./work)



Source Code 로 배포하는 경우 매뉴얼은 필수

- Source Code를 배포하는 오픈 소스마다, 빌드 방법이 모두 다르다.
- 다운로드 페이지 최 하단, 매뉴얼을 찾아 클릭하자.

SmartOS Binaries

Docker Image

Linux on Power LE Systems

Linux on System z

AIX on Power Systems

64-bit

Official Node.js Docker Image

64-bit

64-bit

64-bit

- Signed SHASUMS for release files (How to verify)
- All download options
- Installing Node.js via package manager
- Previous Releases
- Nightly builds
- Unofficial builds
- Building Node.js from source on supported platforms
- Installing Node.js via binary archive

우리는 Unix 시스템이다.

- gcc와 g++, Make, Python 필수
- 메뉴얼에 명시된 대로 설치하자.
 - 코드를 복사해서 터미널에서 실행한다.
- 추가로 distutils 도 설치해준다. (매뉴얼 명시됨)
 - `$sudo apt install python3-distutils -y`

Building Node.js on supported platforms

Note about Python

The Node.js project supports Python ≥ 3 for building and testing.

Unix and macOS

Unix prerequisites

- gcc and g++ ≥ 8.3 or newer
- GNU Make 3.81 or newer
- Python 3.6, 3.7, 3.8, 3.9, or 3.10 (see note above)
 - For test coverage, your Python installation must include pip.

Installation via Linux package manager can be achieved with:

- Ubuntu, Debian: `sudo apt-get install python3 g++ make python3-pip`
- Fedora: `sudo dnf install python3 gcc-c++ make python3-pip`
- CentOS and RHEL: `sudo yum install python3 gcc-c++ make python3-pip`
- OpenSUSE: `sudo zypper install python3 gcc-c++ make python3-pip`
- Arch Linux, Manjaro: `sudo pacman -S python gcc make python-pip`

FreeBSD and OpenBSD users may also need to install `libexecinfo`.

다음 과정을 수행하자.

1. **./work 내에 압축을 풀자**
 - tar.gz 압축 푸는 방법 : 옵션 'J' 대신 'z' 를 사용하면 된다.
→ tar -zxvf [파일명]
2. **압축 툰 디렉토리 이동 후, ./configure 실행**
3. **make -j4**
 - -j4 옵션 : 멀티코어를 사용한 더 빠른 컴파일
 - 약 30 분 소요됨
4. **sudo make install**
 - 1 분 소요됨
5. **node -v**

```
inho@inho-VirtualBox: ~/work/node-v14.16.0
installing /usr/local/include/node/openssl/archs/linux-elf/no
installing /usr/local/include/node/openssl/archs/linux-elf/no
internal/dso_conf.h
installing /usr/local/include/node/openssl/archs/linux-elf/no
internal/bn_conf.h
installing /usr/local/include/node/openssl/archs/linux-elf/no
installing /usr/local/include/node/openssl/archs/linux-elf/no
opensslconf.h
installing /usr/local/include/node/openssl/archs/linux-elf/as
installing /usr/local/include/node/openssl/archs/linux-elf/as
nal/dso_conf.h
installing /usr/local/include/node/openssl/archs/linux-elf/as
nal/bn_conf.h
installing /usr/local/include/node/openssl/archs/linux-elf/as
installing /usr/local/include/node/openssl/archs/linux-elf/as
sslconf.h
installing /usr/local/include/node/zconf.h
installing /usr/local/include/node/zlib.h
inho@inho-VirtualBox:~/work/node-v14.16.0$
inho@inho-VirtualBox:~/work/node-v14.16.0$ node -v
v14.16.0
inho@inho-VirtualBox:~/work/node-v14.16.0$
inho@inho-VirtualBox:~/work/node-v14.16.0$
```

rm 명령어로 작업 내용 삭제

- `rm -r` : 내부 디렉토리 파일까지 모두 삭제
- `rm -rf` : `-f` (force) 옵션 추가, 진짜 지울까요 확인을 무시하고 모두 삭제

모두 제거하는 명령어 (`./work` 디렉토리 삭제)

- `$sudo rm -rf ~/work/*`

dpkg 설치

데비안 패키지 관리 시스템

- 확장자 : .deb
- dpkg 명령어로 설치, 제거, 관리 가능

Raspberry Pi Imager 설치

- 공식사이트 접속하기
 - <https://www.raspberrypi.org/software/>
- 다운로드 URL 복사하기
- 설치 프로그램 다운
- \$wget [URL]

Install Raspberry Pi OS using Raspberry Pi Imager

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi. [Watch our 45-second video](#) to learn how to install an operating system using Raspberry Pi Imager.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

[Download for Ubuntu for x86](#)

[Download for Windows](#)

[Download for macOS](#)



다운로드가 아닌 URL 복사

dpkg 명령어 간단한 예시

- `dpkg -i [경로]` : install 하기
- `dpkg -P [패키지이름]` : 패키지 제거하기
- `dpkg -l` : L 소문자, 설치된 패키지 목록보기

dpkg로 deb 파일 설치

- deb 파일 설치
- `$sudo dpkg -i ./imager_latest_amd64.deb`
- 오른쪽과 같은 메시지가 나오면 성공

```
dpkg: error processing package rpi-imager (--install):  
의존성 문제 - 설정하지 않고 남겨둠  
Processing triggers for gnome-menus (3.36.0-1ubuntu1) ...  
Processing triggers for desktop-file-utils (0.24-1ubuntu3) ...  
Processing triggers for mime-support (3.64ubuntu1) ...  
Processing triggers for hicolor-icon-theme (0.17-2) ...  
처리하는데 오류가 발생했습니다:  
rpi-imager
```

의존성 문제

- 필요한 파일들을 자동으로 설치 해주지 않는다.
- 이 단점을 보완하고자 apt 을 사용한다.

```
dpkg: error processing package rpi-imager (--install):  
의존성 문제 - 설정하지 않고 남겨둠  
Processing triggers for gnome-menus (3.36.0-1ubuntu1) ...  
Processing triggers for desktop-file-utils (0.24-1ubuntu3) ...  
Processing triggers for mime-support (3.64ubuntu1) ...  
Processing triggers for hicolor-icon-theme (0.17-2) ...  
처리하는데 오류가 발생했습니다:  
rpi-imager
```

설치된 패키지 제거

- `$sudo dpkg -P rpi-imager`

편리한 apt install [경로]

- `$sudo apt install ./다운로드 받은 deb 파일`

설치 목록보기

- `$sudo apt list`
- `$sudo apt list | grep rpi-imager`

삭제하기

- `$sudo apt purge rpi-imager`

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?

\$du -h 로 확인

```
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

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.

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 "${[ $? = 0 ]} && echo terminal || echo error" \
"s/^\\s*[0-9]\\+\\s*//;s/[:&]\\s*alert$/\\'''"

# 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.

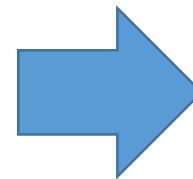
- 예시 2) 자동 세팅 Script 만들기

- 파일명 예시 : setup.sh

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

이 프로그램을 초기화 하고 동작시키려면

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



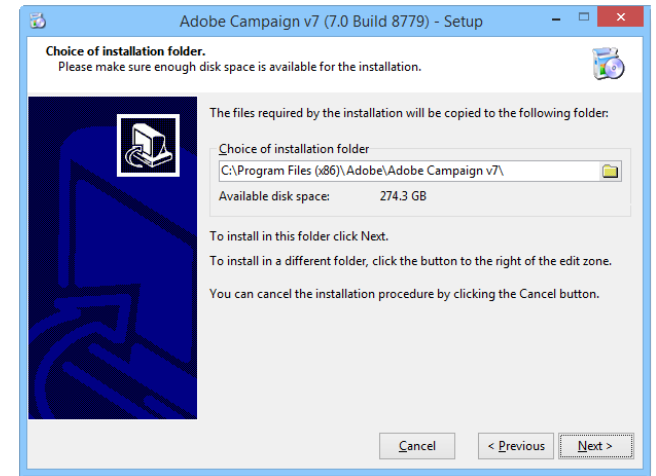
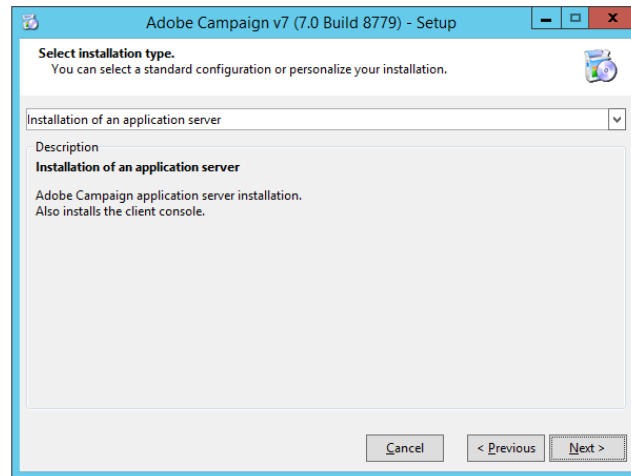
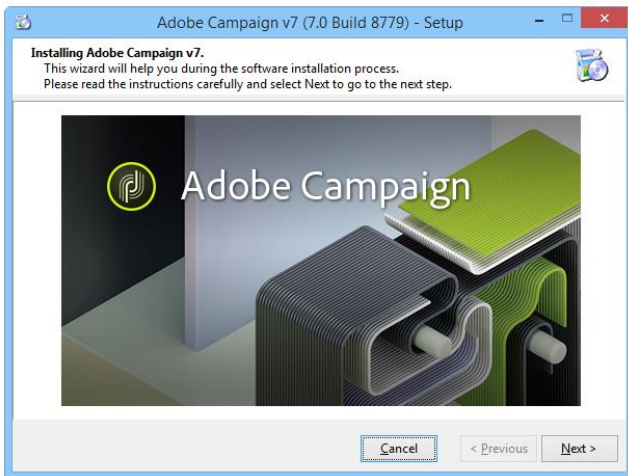
setup.sh
스크립트 파일 생성

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 준비해줘야 한다.
(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.

셸 스크립트 실행

- `$source a.sh`
- `$. a.sh`

a.sh

```
#!/bin/bash  
echo HI
```

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

```
nojin@nojin-VirtualBox:~/d28/dd$ source a.sh
HI
-rw-rw-r-- 1 nojin nojin 0  3월  28 13:04 ./a.txt
-rw-rw-r-- 1 nojin nojin 0  3월  28 13:04 ./b.txt
-rw-rw-r-- 1 nojin nojin 0  3월  28 13:04 ./c.txt
BYE
```

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. (`$sudo chmod a+x ./go.sh`)
- 간헐적으로 사용되는 방법
- 권장 안함 (가독성이 떨어짐)

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
- 실행권한 변경

실행방법 : “**./go.py**”

```
#!/usr/bin/python3
from os import system

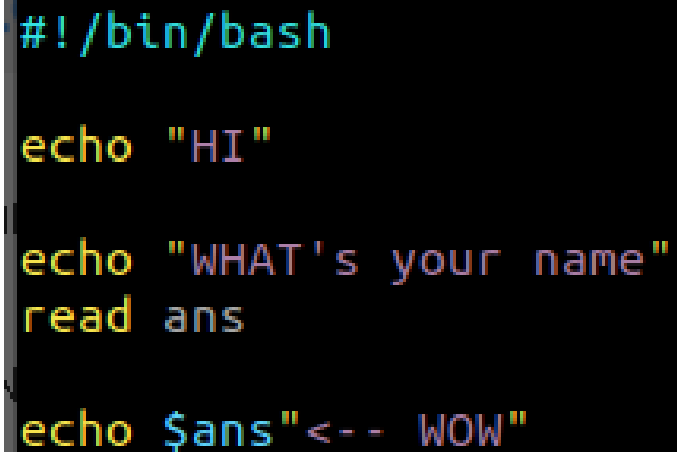
system('echo HI')
```

파이썬 스크립트로
echo 명령어 수행 방법

입/출력

quest.sh 제작하기

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

A terminal window with a red title bar and a black background. The script content is displayed in a monospaced font with syntax highlighting: the shebang is in cyan, echo commands are in yellow, and the read command and its variable are in white. The script is as follows:

```
#!/bin/bash  
  
echo "HI"  
  
echo "WHAT's your name"  
read ans  
  
echo $ans" <- - WOW"
```

변수

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 사용 가능

- \$1
- \$2
- \$3
- ...

```
#!/bin/bash
```

```
echo $1
```

```
echo $2
```

```
echo $3
```

```
nojin@nojin-VirtualBox:~/work$ source quest.sh 100 200 abc  
100  
200  
abc
```

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 ))

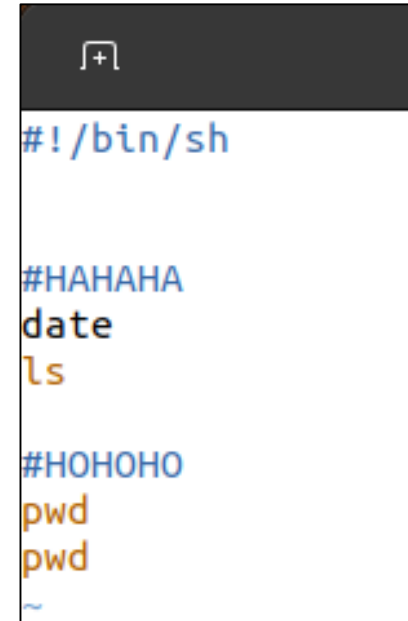
echo $abc
```


다음 스크립트를 제작하자

- **sum.sh 제작하기**
 - Arg 1 : 숫자 1
 - Arg 2 : 숫자 2
 - Arg 3 : 숫자 3
- **출력결과 :**
 - SUM VALUE IS [arg합계] HAHA
- **예시**
 - `$source ./sum.sh 100 200 50`
 - 결과 : SUM VALUE IS 350 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$
```

mem.sh 만들기

- /proc/meminfo 의 MemFree , MemTotal 값을 저장해서 출력한다.

ubuntu 정보 : \$cat /proc/meminfo | grep [정보]

```
nojin@nojin-VirtualBox:~/work$ cat /proc/meminfo | grep MemFree
MemFree:          2559164 kB
nojin@nojin-VirtualBox:~/work$ cat /proc/meminfo | grep MemTotal
MemTotal:         4001372 kB
```

mem.sh 실행 결과

```
nojin@nojin-VirtualBox:~/work$ . mem.sh
meminfo 1: MemTotal:          4001372 kB
meminfo 2: MemFree:           2551328 kB
```

if문

띄어쓰기 조심해야한다.

```
inho@
#!/bin/bash
a=BTS

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

정상 동작



띄어쓰기로 에러

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

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

-lt

- less then

→ a가 50 보다 작으면

- 독보적인 줄임말

```
#!/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 (read 명령어)

- 만약, $10 < x < 30$: “good” 출력
- 아니고 만약, $40 < x < 50$: “omg” 출력
- 그 외 “wow” 출력

```
nojin@nojin-VirtualBox:~/work$ . quest.sh
23
GOOD
nojin@nojin-VirtualBox:~/work$ . quest.sh
45
omg
nojin@nojin-VirtualBox:~/work$ . quest.sh
199
WOW
```

ID 와 PASSWORD 를 정확하게 입력해야 동작하는 프로그램

- secret.sh 파일 생성

- Arg1 : ID 문자열 입력
- Arg2 : PASS 문자열 입력

```
nojin@nojin-VirtualBox:~/work$ . secret.sh KFC 1234
```

- ID 가 “KFC” 이고, 비밀번호가 “1234” 인 경우만 다음 명령어를 수행한다.

- kfc.txt 파일을 생성
- kfc 내용에 “I love my eyes” 내용 입력 (echo)
- 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

```
I love my eyes.  
I love my eyes.  
I love my eyes.  
█
```

```
keep secretn BYE  
nojin@nojin-VirtualBox:~/work$
```

run.sh 쉘 스크립트 제작

- argument 에 Type을 입력 받음

- Type : “dd”

- date 명령어 수행

```
nojin@nojin-VirtualBox:~/work$ . run.sh dd
2023. 02. 08. (수 ) 18:44:13 KST
```

- Type : “sl”

- sl 프로그램 설치 후 실행 (apt install -y)
 - sl 프로그램 삭제 (apt purge -y)

```
nojin@nojin-VirtualBox:~/work$ . run.sh sl
패키지 목록을 읽는 중입니다 ... 완료
의존성 트리를 만드는 중입니다
상태 정보를 읽는 중입니다 ... 완료
다음 패키지가 자동으로 설치되었지만 더 이상 필요
  ubuntu-advantage-desktop-daemon
'sudo apt autoremove'를 이용하여 제거하십시오 .
다음 새 패키지를 설치할 것입니다 :
sl
```

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 개행 문자를 넣어주어야 개행 처리가 된다.
- 예시
 - printf "HI HI \n"
 - printf "NAME : %s \n" "INHO"

```
#!/bin/bash
```

```
a=100
```

```
b=3.14
```

```
c="BTS"
```

```
printf "HI %d H0 %f C0 %s \n" $a $b $c
```

```
noj in@noj in-VirtualBox:~/work$ . quest.sh  
HI 100 H0 3.140000 C0 BTS
```


배열 만들기

- 배열 만들 때는 **()** 를 사용
- 배열 값을 출력할 때는 **{}** 괄호 필수

```
#!/bin/bash

arr=( 10,20,30 )

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

```
nojin@nojin-VirtualBox:~/work$ . quest.sh
10,20,30
```

for 문법 비슷하다

```
#!/bin/bash

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

```
#!/bin/bash

arr=(10,20,30)
for ((i=0; i<3; i++))
do
    echo ${arr[i]}
done
```

함수 만들기

- 함수 호출시, 함수 이름만 입력하면 된다.

```
#!/bin/bash

abc(){
    printf "HIIH\n"
}

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.

```
hen
    . /usr/share/bash-completion/bash_completion
elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi
fi
export KFC=HELLO
```

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.

환경변수로 등록하면, 어디서든 사용 가능하다!
아래와 같이 스크립트 내에서도 사용 가능하다

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

```
nojin@nojinVx:~/tets$ . quest.sh  
HELLO World
```

Hello.sh 만들기

- 만약 환경변수 값 KFC 가 HELLO 값을 가진다면,
 - OH GOOD HI 를 출력한다.
- 그렇지 않으면,
 - OH MY GOD 을 출력한다.

```
noj in@noj inVx:~/tets$ . Hello.sh  
OH GOOD HI
```

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@inhop: ~  
# 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.xz
- 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