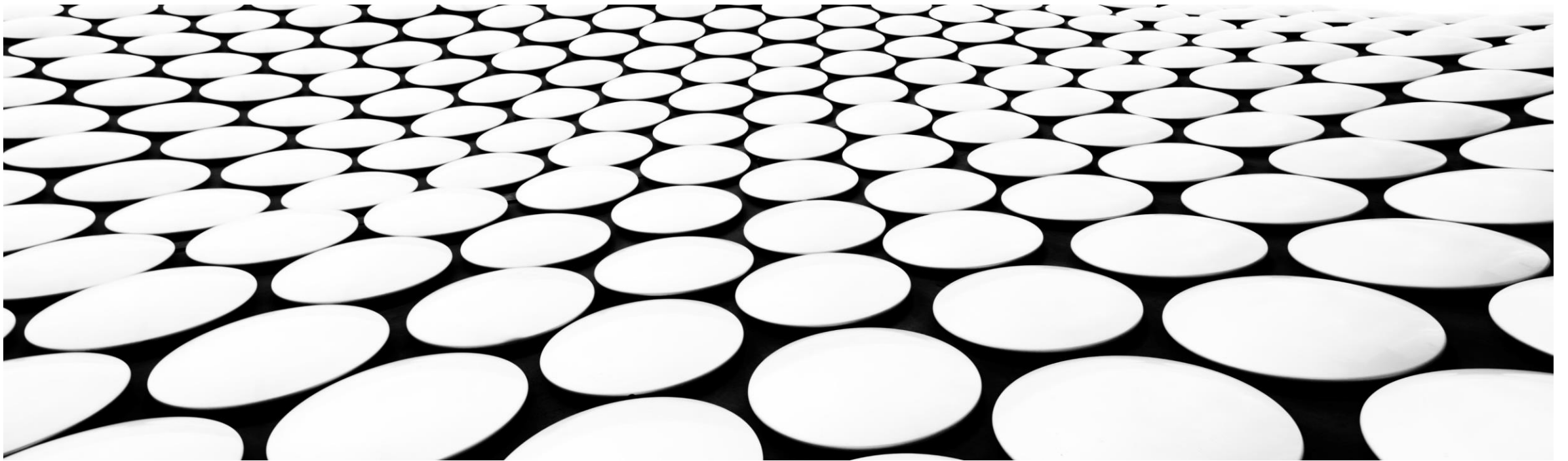
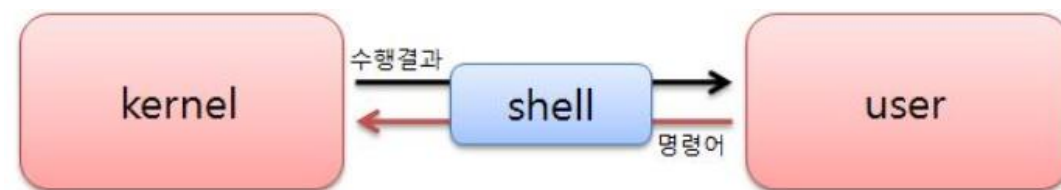

LINUX의 셸 스크립트 및 MAKEFILE



LINUX - SHELL SCRIPT

Shell 이란?

- 셸(Shell)은 커널(Kernel)과 사용자간의 다리 역할을 하는 것으로 사용자로부터 명령을 받아 그것을 해석하고 프로그램을 실행하는 역할을 한다.
- 셸은 사용자가 시스템에 로그인하게 되면 각 사용자에게 설정된 셸이 부여 되면서 다양한 명령어를 수행할 수 있게 된다. 달리 말하면 사용자에게 셸을 부여하지 않게 되면 시스템에 로그인하더라도 명령을 수행할 수 없게 되므로 로그인을 막는 효과와 동일하다고 볼 수 있다.



LINUX - SHELL SCRIPT

셸의 역사

- 리눅스의 모태가 되는 유닉스 최초의 셸은 켄 톰프슨(Ken Thompson)이 멀틱스(Multics) 셸을 따라 모형화한 셸을 이용하였고, 그 후 스티븐 본(Steven Bourne)이 유닉스 버전 7의 기본 셸이 되는 본 셸(Bourne Shell, sh)을 개발하였다. 본 셸은 강력한 셸이었지만 유용한 기능이 많지 않았다.
- 그 후, 버클리 대학의 빌 조이(Bill Joy)가 개발한 C 셸(C shell, csh)이 등장하였으며 현재에는 bash, ksh, tcsh, zsh과 같은 다양한 셸이 개발 되었다. 리눅스에는 sh를 기본으로 ksh와 csh 계열의 장점을 결합한 bash shell(Bourne Again shell)을 표준으로 하고 있다.

LINUX - SHELL SCRIPT

셸의 종류

■ bourne 셸 계열의 셸

- 1. sh (bourne shell)
- 가장 기본적인 셸로 유닉스의 초기부터 사용되어 온 셸이다. 스크립트를 지원한다.
- 2. ksh (korn shell)
- 본 셸을 확장한 셸이다. 본 셸의 명령어를 모두 인식하며, 명령어 히스토리(history) 기능과 앨리어스(alias), 작업 제어 등의 기능이 추가되었다. 일반적으로 유닉스에서 가장 많이 사용되는 셸이다. 명령행 편집기능을 제공한다.
- 3. bash (Bourne Again Shell)
- 리눅스에서 가장 많이 사용하는 셸이다. C 셸과 콘 셸의 장점을 결합하여 작성되었으며, Bourne 셸 문법의 명령어 셋을 제공하여 Bourne Shell과 호환되는 셸로 GNU 프로젝트에 의해 만들어지고 배포된다. 명령행 편집기능을 제공한다.

■ C 셸 계열의 셸

- 1. csh (C Shell)
- 명령행 편집기능을 제공하지 않는다. C 언어 위주의 셸로 처음 작성되었을 때에는 본 셸이 가지고 있지 못한 기능들(작업제어, 명령어 히스토리 등)을 가지고 있었기 때문에 많이 사용되었다.
- 2. tcsh (TC Shell)
- csh의 기능을 강화한 셸이다. 확장 C Shell. 명령행 편집 기능을 제공한다.
- CentOS, SUSE Linux, Asianux 등의 레드햇 계열의 리눅스 배포판에서 기본으로 사용되지 않는 셸
- Zsh 셸
- 로그인셸 및 셸스크립트 명령어 프로세서로서 이용 가능한 유닉스 셸이다. 표준 셸들 중에서 zsh는 ksh와 가장 유사하지만 많은 개선들을 포함한다. Zsh는 명령행 편집, 내장 스펠링 수정, history 등의 기능을 가진다.
- Ash 셸
- 추가적인 기능들이 없이 본 셸에 가장 부합하는 셸이다. 본 셸은 상업적인 유닉스 시스템들에서 사용 가능하므로, ash는 셸 스크립트가 본 셸에 잘 부합하는지 시험할 때 유용하다. 또한 이것은 다른 sh- 호환 셸에 비해 적은 메모리와 공간을 요구한다.

LINUX - SHELL SCRIPT

bash shell 이란

- bash 셸은 1989년 브라이언 폭스(Brian Fox)가 GNU 프로젝트를 위해 개발하였으며 본 셸(Bourne Shell)을 기반으로 만들어졌다.
- GNU 운영체제, 리눅스, 맥OS X 등 다양한 운영체제에서 사용 중이며 현재 리눅스의 표준 셸이다. bash의 명령어 문법은 sh와 호환되고 ksh와 csh의 유용한 기능을 참고하여 명령 히스토리, 명령어 완성 기능, 히스토리 치환, 명령행 편집 등을 지원하고 있다.

LINUX - SHELL SCRIPT

bash shell 예약 변수

- 쉘 스크립트에서 사용자가 정해서 만들 수 없는 이미 정의된 변수가 존재한다. 그것을 예약 변수라고 한다. 몇가지 예약 변수를 보도록 합시다.
- Linux의 shell은 xterm에서 구동된다.

변수	설명
HOME	사용자 홈 디렉토리를 의미한다.
PATH	실행 파일의 경로이다. 여러분이 chmod, mkdir 등의 명령어들은 /bin이나 /usr/bin, /sbin에 위치하는데, 이 경로들을 PATH 지정하면 여러분들은 굳이 /bin/chmod를 입력하지 않고, chmod 입력만 해주면 된다.
LANG	프로그램 실행 시 지원되는 언어를 말한다.
UID	사용자의 UID이다.
SHELL	사용자가 로그인시 실행되는 쉘을 말한다.
USER	사용자의 계정 이름을 말한다.
FUNCNAME	현재 실행되고 있는 함수 이름을 말한다.
TERM	로그인 터미널을 말한다.

LINUX - SHELL SCRIPT

bash shell 프롬프트 구성

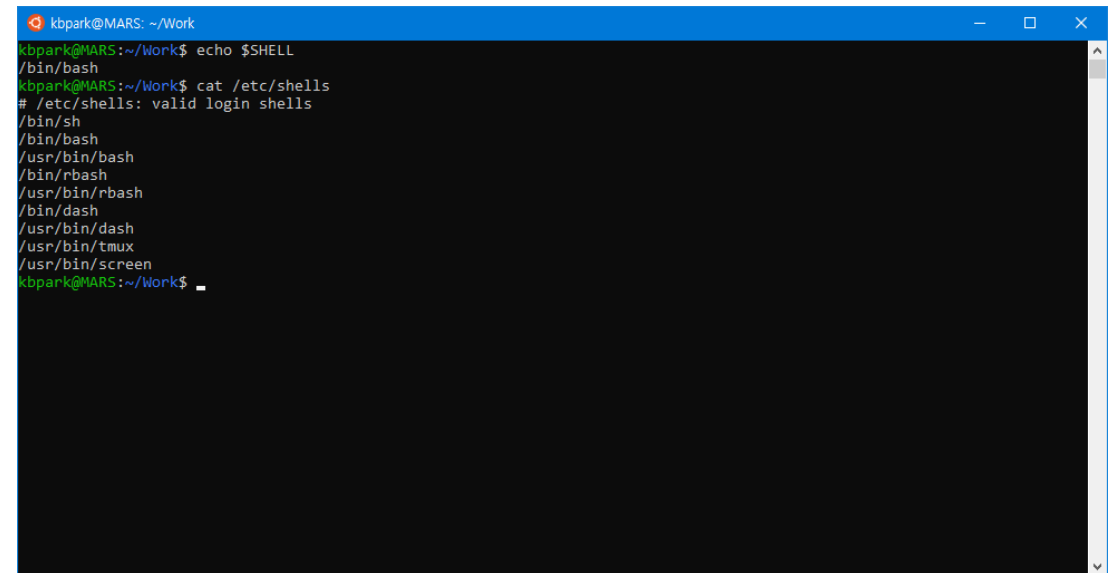
- 리눅스환경에서는 사용자마다 다른 셸을 지정할 수 있다. bash shell에 접속하게 된다면 일반적으로 보게 될 화면을 다음과 같다.

```
kbpark@MARS: ~/Work
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106:/:nonexistent:/usr/sbin/nologin
syslog:x:104:110:/home/syslog:/usr/sbin/nologin
_apt:x:105:65534:/:nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuid:x:107:112:/run/uuid:/usr/sbin/nologin
tcpdump:x:108:113:/:nonexistent:/usr/sbin/nologin
sshd:x:109:65534:/run/ssh:/usr/sbin/nologin
landscape:x:110:115:/var/lib/landscape:/usr/sbin/nologin
pollinate:x:111:1:/var/cache/pollinate:/bin/false
kbpark:x:1000:1000:,,,:/home/kbpark:/bin/bash
kbpark@MARS:~$ ls
Download Work litecoin-api-exam.tar.gz
kbpark@MARS:~$ cd Work/
kbpark@MARS:~/Work$ ls
bitcoin facebook_react litecoin litecoin_0.13 litecoin_0.15 makecoin node_exam shell_exam solidity_exam
kbpark@MARS:~/Work$ ls -al
total 44
drwxr-xr-x 11 kbpark kbpark 4096 May 18 11:15 .
drwxr-xr-x 16 kbpark kbpark 4096 May 25 21:09 ..
drwxr-xr-x 14 kbpark kbpark 4096 Apr 1 10:35 bitcoin
drwxr-xr-x 9 kbpark kbpark 4096 May 18 11:16 facebook_react
drwxr-xr-x 14 kbpark kbpark 4096 Mar 18 11:05 litecoin
drwxr-xr-x 11 kbpark kbpark 4096 Mar 18 11:07 litecoin_0.13
drwxr-xr-x 12 kbpark kbpark 4096 Mar 18 11:08 litecoin_0.15
drwxr-xr-x 8 kbpark kbpark 4096 May 7 10:42 makecoin
drwxr-xr-x 4 kbpark kbpark 4096 May 14 11:34 node_exam
drwxr-xr-x 2 kbpark kbpark 4096 May 7 10:53 shell_exam
drwxr-xr-x 6 kbpark kbpark 4096 Apr 9 17:57 solidity_exam
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

셸의 확인

- 시스템에 로그인한 후에 사용 중인 셸을 확인하려면 로그인 셸 관련 환경변수 SHELL을 이용해서 가능하다. 즉 아래와 같은 명령문으로 확인할 수 있다.
 - `$ echo $SHELL`
- 시스템에 사용 가능한 셸의 리스트를 확인 하려면 다음과 같은 명령문으로 확인할 수 있다.
 - `$ cat /etc/shells`



```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ echo $SHELL
/bin/bash
kbpark@MARS:~/Work$ 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
/usr/bin/tmux
/usr/bin/screen
kbpark@MARS:~/Work$
```


LINUX - SHELL SCRIPT

셸의 변경

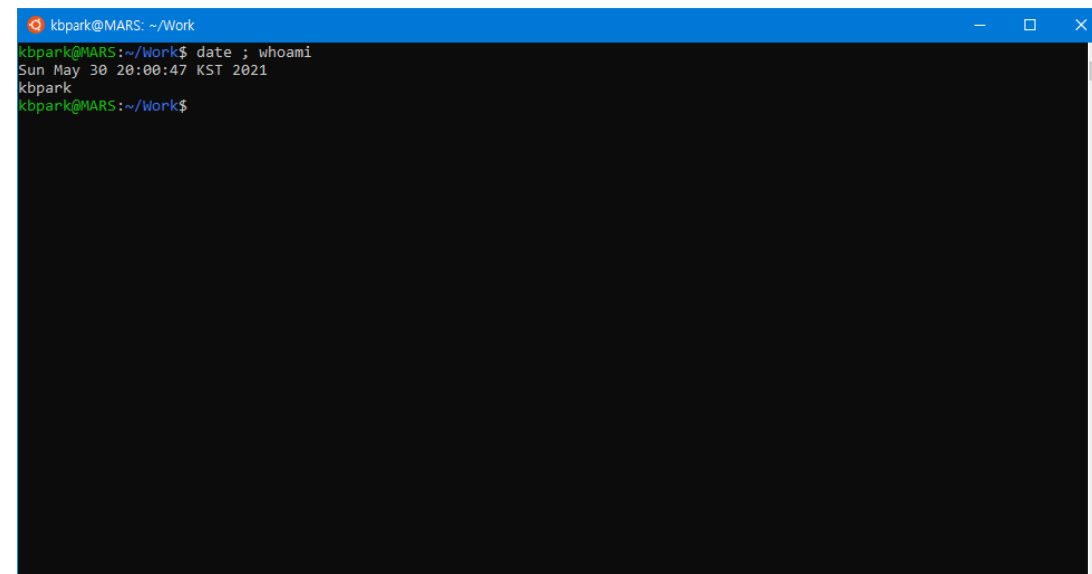
- 셸 리스트 중 현재와 다른 셸을 사용하려면 'chsh'라는 명령어를 이용하여 셸의 절대경로를 입력하면 된다.
- 변경한 셸은 다음 로그인부터 적용된다.
- 다음은 sh 셸에서 bash 셸로 바꾼 예이다.
- 최근 mac은 zsh 셸을 사용 하고 있다.

```
# echo $SHELL
/bin/sh
# chsh
Changing the login shell for root
Enter the new value, or press ENTER for the default
    Login Shell [/bin/sh]: /bin/bash
#
test1@ServerTest2:~$ su -
Password:
root@ServerTest2:~# echo $SHELL
/bin/bash
root@ServerTest2:~#
```

LINUX - SHELL SCRIPT

여러 명령 사용

- 세미콜론(;)
 - 하나의 라인에 주어진 명령어들을 성공,실패와 관련 없이 전부 실행한다.
 - \$ 명령1 ; 명령2 ; 명령3 ;
 - Ex : \$ date ; who

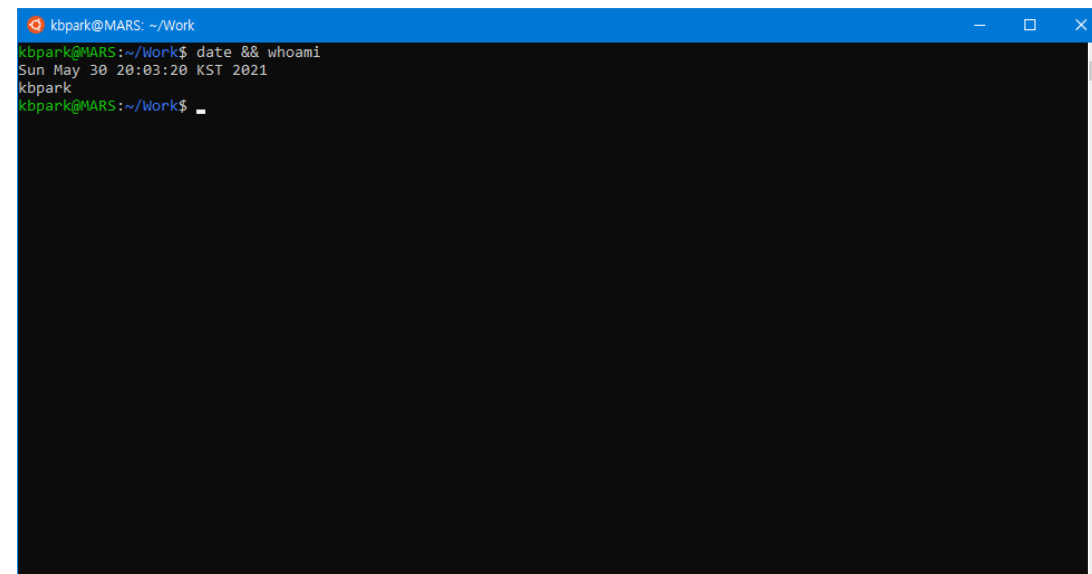
A terminal window titled 'kbpark@MARS: ~/Work' with a blue header bar. The terminal shows the command 'date ; whoami' being executed. The output is 'Sun May 30 20:00:47 KST 2021' followed by the username 'kbpark' on the next line. The prompt 'kbpark@MARS:~/Work\$' is visible at the bottom.

```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ date ; whoami
Sun May 30 20:00:47 KST 2021
kbpark
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

여러 명령 사용

- 앰퍼샌드(&&)
 - 앞에서부터 순차적으로 실행하되, 명령 실행에 실패할 경우 뒤에 오는 명령은 실행하지 않는다.
 - \$ 명령1 && 명령2 && 명령3 &&
 - Ex : \$ date && who
- Congratulation 여러분은 지금 쉘 스크립트를 작성 했다.
- 한 줄에 입력할 수 있는 명령어의 한계는 255자이다.
- 이렇게 스크립트를 실행 하려면 번번히 명령어 라인에 기입 해야 하는데 이것을 파일에 저장 해서 실행 해야 한다.

A terminal window titled 'kbpark@MARS: ~/Work' with a blue header bar. The terminal shows a sequence of commands and their outputs. The first command is 'date && whoami', which outputs 'Sun May 30 20:03:20 KST 2021' followed by the username 'kbpark'. The second command is 'kbpark', which outputs 'kbpark'. The prompt returns to '\$' after each command.

```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ date && whoami
Sun May 30 20:03:20 KST 2021
kbpark
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

셸 스크립트 파일 작성하기

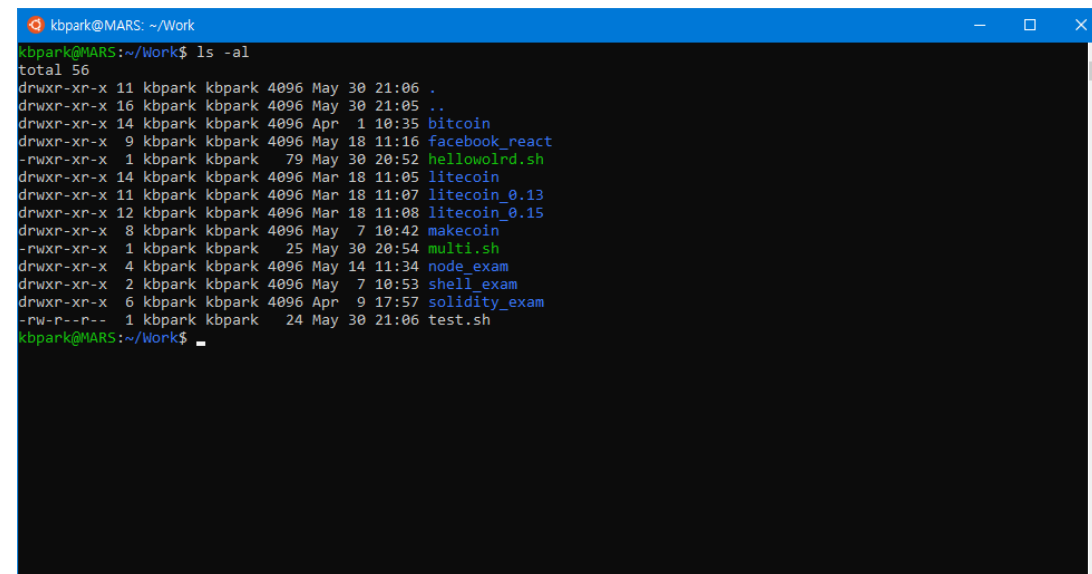
- 이제 첫 번째는 bash 셸이 스크립트 파일을 인식하는 것이다.
 - 셸은 명령을 찾기 위해 PATH라는 환경 변수에 입력하고 실행할 수 있도록 하는 것이다.
- 두 번째는 절대 경로에서 실행 하는 것이다.
 - `$./multicommand.sh`
 - `Bash: ./ multicommand.sh: Permission denied`

- `$ data ; whoami multicommand.sh` 스크립트
 - `#!/bin/bash`
 - `date`
 - `whoami`

LINUX - SHELL SCRIPT

셸 스크립트 파일 작성하기

- 스크립트를 작성 했지만 보는 바와 같이 권한 에러가 난다.
 - 실행 권한이 없어 실행이 안되는 것이다.



```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ ls -al
total 56
drwxr-xr-x 11 kbpark kbpark 4096 May 30 21:06 .
drwxr-xr-x 16 kbpark kbpark 4096 May 30 21:05 ..
drwxr-xr-x 14 kbpark kbpark 4096 Apr  1 10:35 bitcoin
drwxr-xr-x  9 kbpark kbpark 4096 May 18 11:16 facebook_react
-rwxr-xr-x  1 kbpark kbpark  79 May 30 20:52 helloworld.sh
drwxr-xr-x 14 kbpark kbpark 4096 Mar 18 11:05 litecoin
drwxr-xr-x 11 kbpark kbpark 4096 Mar 18 11:07 litecoin_0.13
drwxr-xr-x 12 kbpark kbpark 4096 Mar 18 11:08 litecoin_0.15
drwxr-xr-x  8 kbpark kbpark 4096 May  7 10:42 makecoin
-rwxr-xr-x  1 kbpark kbpark  25 May 30 20:54 multi.sh
drwxr-xr-x  4 kbpark kbpark 4096 May 14 11:34 node_exam
drwxr-xr-x  2 kbpark kbpark 4096 May  7 10:53 shell_exam
drwxr-xr-x  6 kbpark kbpark 4096 Apr  9 17:57 solidity_exam
-rw-r--r--  1 kbpark kbpark  24 May 30 21:06 test.sh
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

셸 스크립트 파일 작성하기

- 실행 권한 주기
 - chmod
 - \$ chmod 755 multicommand.sh
 - .\$ /multicommand.sh

```
kbpark@MARS: ~/Work
drwxr-xr-x 14 kbpark kbpark 4096 Mar 18 11:05 litecoin
drwxr-xr-x 11 kbpark kbpark 4096 Mar 18 11:07 litecoin_0.13
drwxr-xr-x 12 kbpark kbpark 4096 Mar 18 11:08 litecoin_0.15
drwxr-xr-x 8 kbpark kbpark 4096 May 7 10:42 makecoin
-rwxr-xr-x 1 kbpark kbpark 25 May 30 20:54 multi.sh
drwxr-xr-x 4 kbpark kbpark 4096 May 14 11:34 node_exam
drwxr-xr-x 2 kbpark kbpark 4096 May 7 10:53 shell_exam
drwxr-xr-x 6 kbpark kbpark 4096 Apr 9 17:57 solidity_exam
-rw-r--r-- 1 kbpark kbpark 24 May 30 21:06 test.sh
kbpark@MARS:~/Work$ chmod 755 test.sh
kbpark@MARS:~/Work$ ls -al
total 56
drwxr-xr-x 11 kbpark kbpark 4096 May 30 21:06 .
drwxr-xr-x 16 kbpark kbpark 4096 May 30 21:05 ..
drwxr-xr-x 14 kbpark kbpark 4096 Apr 1 10:35 bitcoin
drwxr-xr-x 9 kbpark kbpark 4096 May 18 11:16 facebook_react
-rwxr-xr-x 1 kbpark kbpark 79 May 30 20:52 helloworld.sh
drwxr-xr-x 14 kbpark kbpark 4096 Mar 18 11:05 litecoin
drwxr-xr-x 11 kbpark kbpark 4096 Mar 18 11:07 litecoin_0.13
drwxr-xr-x 12 kbpark kbpark 4096 Mar 18 11:08 litecoin_0.15
drwxr-xr-x 8 kbpark kbpark 4096 May 7 10:42 makecoin
-rwxr-xr-x 1 kbpark kbpark 25 May 30 20:54 multi.sh
drwxr-xr-x 4 kbpark kbpark 4096 May 14 11:34 node_exam
drwxr-xr-x 2 kbpark kbpark 4096 May 7 10:53 shell_exam
drwxr-xr-x 6 kbpark kbpark 4096 Apr 9 17:57 solidity_exam
-rwxr-xr-x 1 kbpark kbpark 24 May 30 21:06 test.sh
kbpark@MARS:~/Work$ ./multi.sh
Sun May 30 21:13:38 KST 2021
kbpark
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

셸 스크립트 파일 작성하기

- 셸 스크립트는 셸에게 무슨 명령들을 실행할지 알려주는 스크립트 파일이다. 우리는 가장 널리 쓰이는 bash 셸을 사용하는 스크립트를 설명하도록 한다.
- `#!/bin/bash`
- 스크립트 최 상단에는 항상 이 구문이 적혀 있어야한다. 간단하게 hello, world라는 문자열을 출력하는 스크립트를 만들어봅시다. 파일명은 helloworld.sh로 한다.
 - `#!/bin/bash`
 - `echo "hello, world"`
 - `printf "hello, world"`

LINUX - SHELL SCRIPT

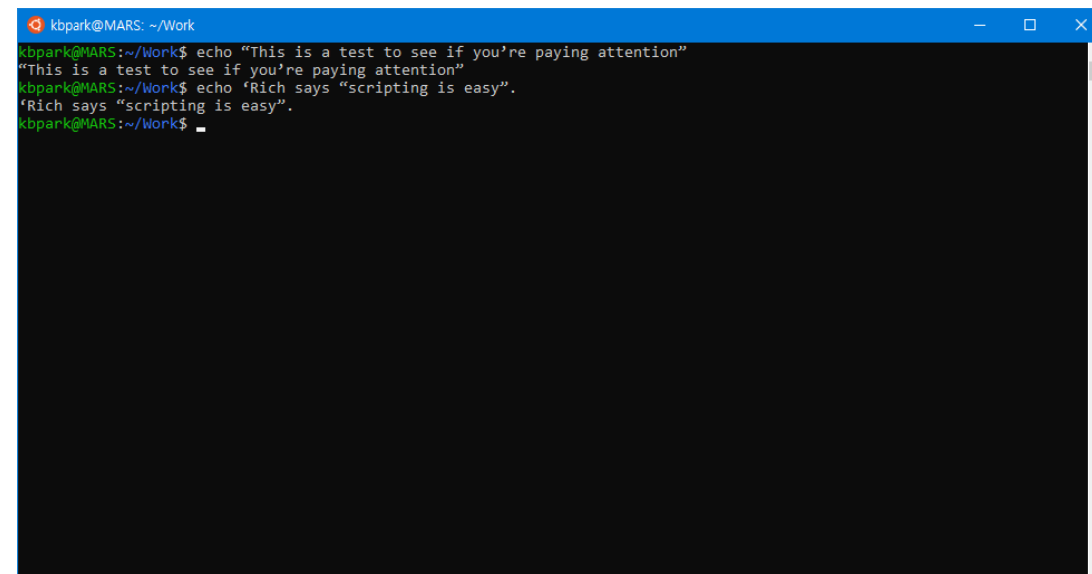
메시지 표시하기

- 대부분의 셸 명령은 스크립트가 실행될 때 콘솔 모니터에 나름대로의 출력을 표시한다. 하지만 스크립트 사용자가 스크립트 안에서 무슨 일이 일어나고 있는지 알 수 있도록 별도의 문자 메시지를 추가하고 싶을 때가 자주 있을 것이다.
- echo 명령을 이용하면 이러한 일을 할 수 있다.
- echo 명령 뒤에 문자열을 추가 하면 텍스트 문자열을 표시할 수 있다.
 - `$ echo This is a test`
 - `This is a test`
 - `$`

LINUX - SHELL SCRIPT

메시지 표시하기

- 표시하고자 하는 문자열을 나타내기 위해 따옴표(') 따로 사용하지 않아도 된다.
- 문자열 안에서 따옴표는 가끔 이상한 결과가 나온다.
 - `$ echo Let's see if this'll work`
 - `Lets see if thisll work`
- `echo` 명령은 텍스트 문자열을 묶기 위해서 홑따옴표 또는 겹따옴표를 쓴다. 문자열 안에서 따옴표를 사용하면 텍스트 안에서 한 가지 유형의 따옴표를 사용한 다음 문자열을 묶을 때에는 다른 유형의 따옴표를 써야 한다.
 - `$ echo "This is a test to see if you're paying attention"`
 - `This is a test to see if you're paying attention`
 - `$ echo 'Rich says "scripting is easy".'`
 - `Rich says "scripting is easy".`

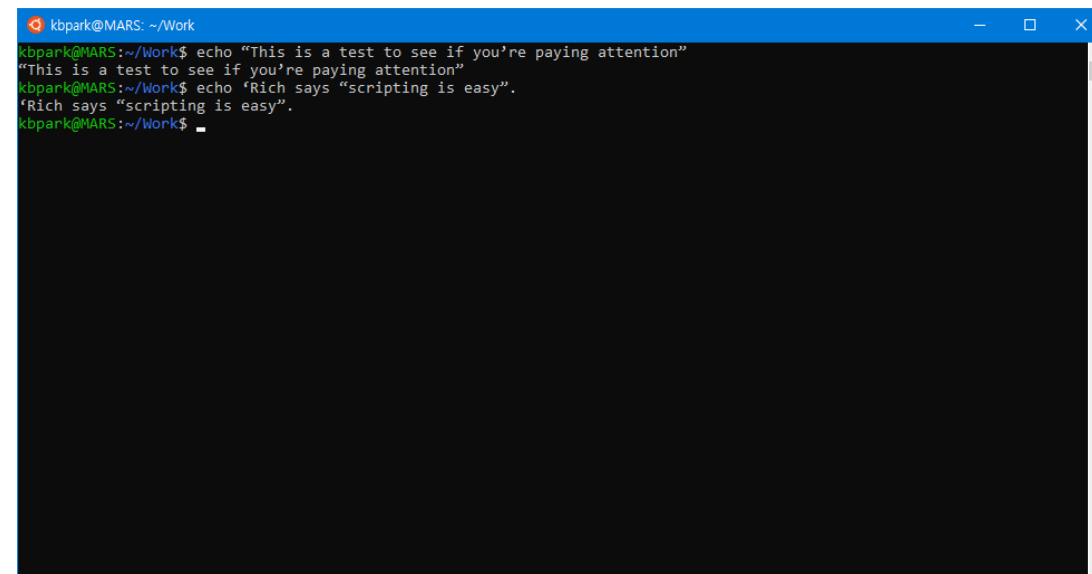


```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ echo "This is a test to see if you're paying attention"
This is a test to see if you're paying attention
kbpark@MARS:~/Work$ echo 'Rich says "scripting is easy".'
Rich says "scripting is easy".
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

메시지 표시하기

- 표시하고자 하는 문자열을 나타내기 위해 따옴표(') 따로 사용하지 않아도 된다.
- 문자열 안에서 따옴표는 가끔 이상한 결과가 나온다.
 - `$ echo Let's see if this'll work`
 - `Lets see if thisll work`
- `echo` 명령은 텍스트 문자열을 묶기 위해서 홑따옴표 또는 겹따옴표를 쓴다. 문자열 안에서 따옴표를 사용하면 텍스트 안에서 한 가지 유형의 따옴표를 사용한 다음 문자열을 묶을 때에는 다른 유형의 따옴표를 써야 한다.
 - `$ echo "This is a test to see if you're paying attention"`
 - `This is a test to see if you're paying attention`
 - `$ echo 'Rich says "scripting is easy".'`
 - `Rich says "scripting is easy".`

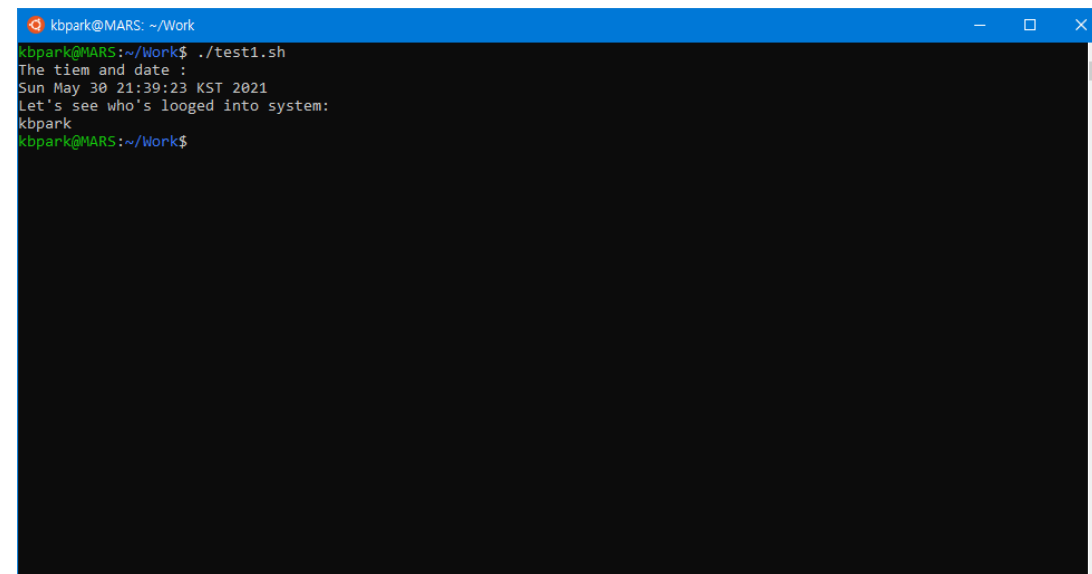


```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ echo "This is a test to see if you're paying attention"
This is a test to see if you're paying attention
kbpark@MARS:~/Work$ echo 'Rich says "scripting is easy".'
Rich says "scripting is easy".
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

메시지 표시하기

- touch test1.sh
- vi test1.sh
 - #!/bin/bash/
 - #This is test shell script
 - echo The time and date are:
 - Date
 - echo "Let's see who's logged into the system:"
 - whoami

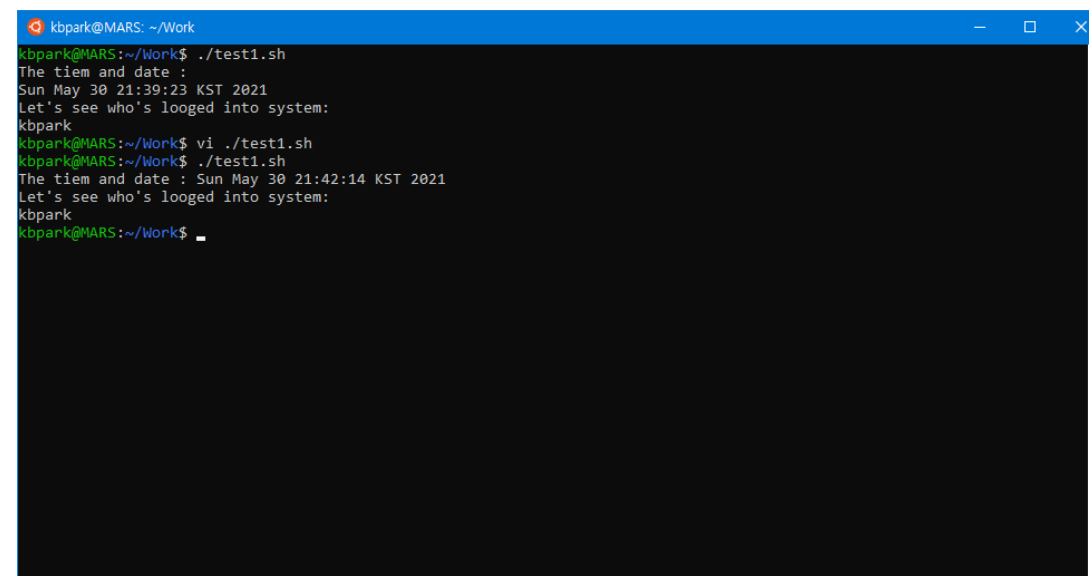
A terminal window titled 'kbpark@MARS: ~/Work' showing the execution of a shell script. The prompt is 'kbpark@MARS:~/Work\$./test1.sh'. The script output is: 'The time and date :', 'Sun May 30 21:39:23 KST 2021', 'Let's see who's logged into system:', 'kbpark', and the prompt returns to 'kbpark@MARS:~/Work\$'.

```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ ./test1.sh
The time and date :
Sun May 30 21:39:23 KST 2021
Let's see who's logged into system:
kbpark
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

메시지 표시하기

- touch test1.sh
- vi test1.sh
 - #!/bin/bash/
 - #This is test shell script
 - echo -n "The time and date are:"
 - date
 - echo "Let's see who's logged into the system:"
 - whoami

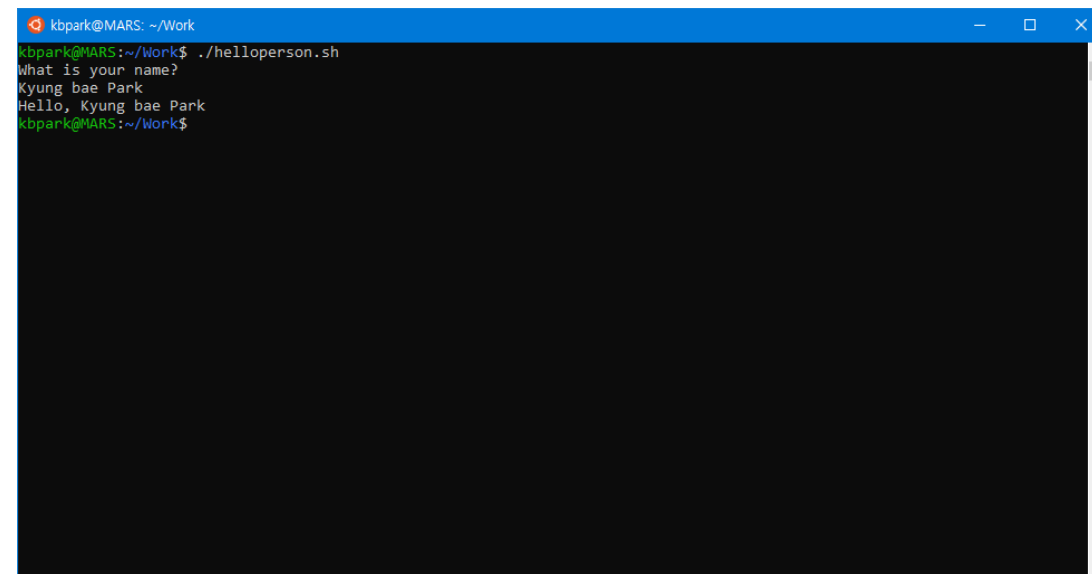


```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ ./test1.sh
The tiem and date :
Sun May 30 21:39:23 KST 2021
Let's see who's looged into system:
kbpark
kbpark@MARS:~/Work$ vi ./test1.sh
kbpark@MARS:~/Work$ ./test1.sh
The tiem and date : Sun May 30 21:42:14 KST 2021
Let's see who's looged into system:
kbpark
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

Extended shell script

- 셸 스크립트를 수행할 작업과 시기를 알려주는 필수 구성요소가 필요하다. 대부분의 셸 스크립트는 이보다 복잡하지만, 셸 스크립트 역시 일종의 프로그래밍 언어이며, 변수, 함수, 제어, 반복 등과 같은 구조로 이루어진다. 스크립트가 복잡한 구조로 되어 있어도 순차적인 실행 구조를 가진다.
- 다음은 간단한 입력 구조를 가지는 셸 스크립트 예제이다.
 - `#!/bin/bash`
 - `echo "What is your name?"`
 - `read PERSON`
 - `echo "Hello, $PERSON"`

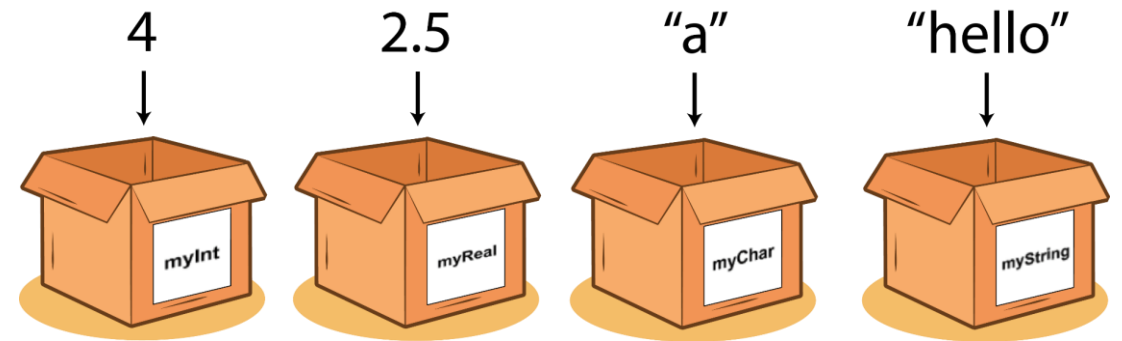
A terminal window titled 'kbpark@MARS: ~/Work' showing the execution of a script named 'helloperson.sh'. The script prompts 'What is your name?', the user enters 'Kyung bae Park', and the script outputs 'Hello, Kyung bae Park'. The prompt returns to 'kbpark@MARS:~/Work\$'.

```
kbpark@MARS: ~/Work
kbpark@MARS:~/Work$ ./helloperson.sh
What is your name?
Kyung bae Park
Hello, Kyung bae Park
kbpark@MARS:~/Work$
```

LINUX - SHELL SCRIPT

Using shell variable

- 셸에서 변수를 사용하는 방법에 대해 알아본다. 변수는 "아직 알려지지 않거나 어느 정도까지만 알려져 있는 양이나 정보에 대한 상징적인 이름이다. 컴퓨터 소스 코드에서의 변수 이름은 일반적으로 데이터 저장 위치와 그 안의 내용물과 관련되어 있으며 이러한 것들은 프로그램 실행 도중에 변경될 수 있다."
- 이를 쉽게 정의하면 다음과 같다.
 - 변수는 컴퓨터 메모리에 존재한다.
 - 할당된 메모리 공간은 정보를 저장하기 위해서 사용된다.
 - 정보가 저장된 공간을 찾기 위해서, 이름을 붙여서 사용한다.
- 변수의 할당된 값은 숫자, 텍스트 파일, 파일 이름, 장치 또는 다른 유형의 데이터일 수 있으며, 변수는 할당된 메모리의 주소를 나타내는 포인터이기 때문에 변수를 생성, 할당, 삭제가 가능하다.



LINUX - SHELL SCRIPT

Variable Names

- 셸에서 변수 이름을 지칭하는 규칙은 다음과 같다.
 - 변수 안에 들어갈 수 있는 글자는 a to z, A to Z이다.
 - 변수 안에 들어갈 수 있는 숫자는 0 ~ 9까지 이다.
 - 서로 다른 변수 이름을 이어서 사용하기 원한다면 underscore character (_)을 사용한다.
 - 셸 변수의 이름은 대문자를 사용한다.

LINUX - SHELL SCRIPT

Variable Names

- 올바른 변수 선언의 예제를 살펴보면 다음과 같다.
 - _ALL
 - NAME
 - VAR_1
 - VAR_2
- 잘못된 변수 선언의 예제는 다음과 같다.
 - 2_VAR
 - -VARIABLE
 - VAR1-VAR2
 - VAR_A!

셸에서!, -, *와 같은 특수문자를 사용할 수 없는 이유는 셸 자체에서 지칭하는 의미가 존재하기 때문이다.

LINUX - SHELL SCRIPT

Defining Variables

- 변수를 정의하는 일반적인 방법은 다음과 같다.
 - `variable_name=variable_value`
 - `NAME="Lucas"`
- 예제에서 확인할 수 있듯이 NAME이라는 변수를 정의하고 "Lucas"라는 값을 대입할 수 있다. 이러한 유형의 변수를 **스칼라** 변수라고 지칭하는데 스칼라 변수는 한 번에 하나의 값을 저장할 수 있다는 특징을 가지고 있다.
 - `ORGANIZATIONS="wisoft"`
 - `NUMER_OF_PEOPLE=30`

LINUX - SHELL SCRIPT

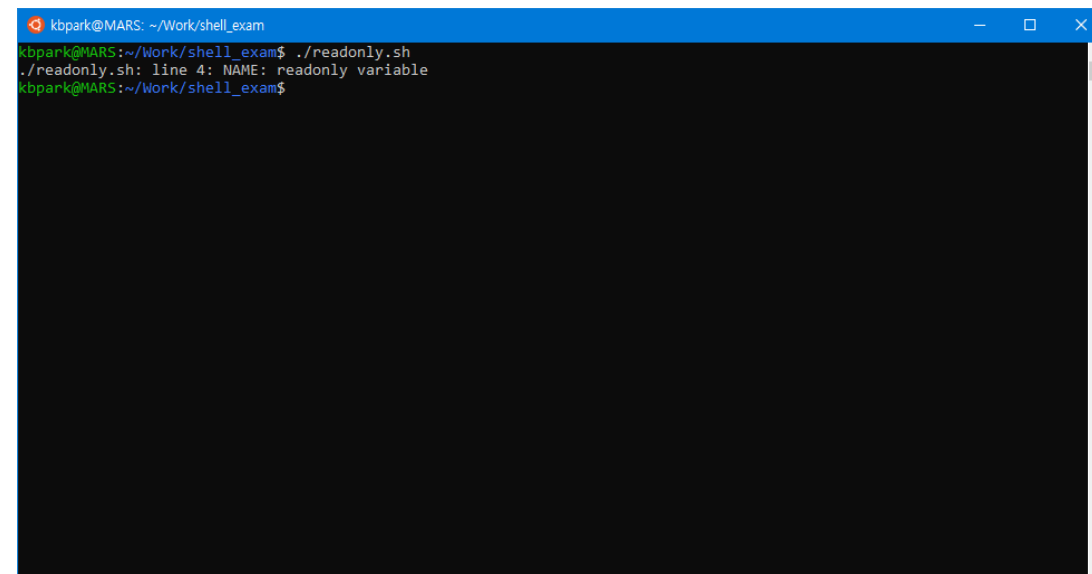
Accessing Values

- 변수에 저장된 값에 접근하기 위해서는 이름 앞에 \$기호를 붙여야 한다. 다음 예제는 정의된 변수 NAME 값에 접근하고 **STDOUT**으로 출력한다.
 - `#!/bin/bash`
 - `NAME="Lucas"`
 - `echo $NAME`

LINUX - SHELL SCRIPT

Read-only Variables

- 셸에서는 읽기 전용으로 변수를 지정할 수 있으며, 읽기 전용으로 지정된 변수는 값을 변경할 수 없다.
- 다음 예제는 읽기 전용 변수에서 값을 변경하고자 할 때 나타나는 에러를 보여준다.
 - `#!/bin/bash`
 - `NAME="Lucas"`
 - `readonly NAME`
 - `NAME="Kyung Bae Park"`
- `$./readonly.sh: line 4: NAME: readonly variable`

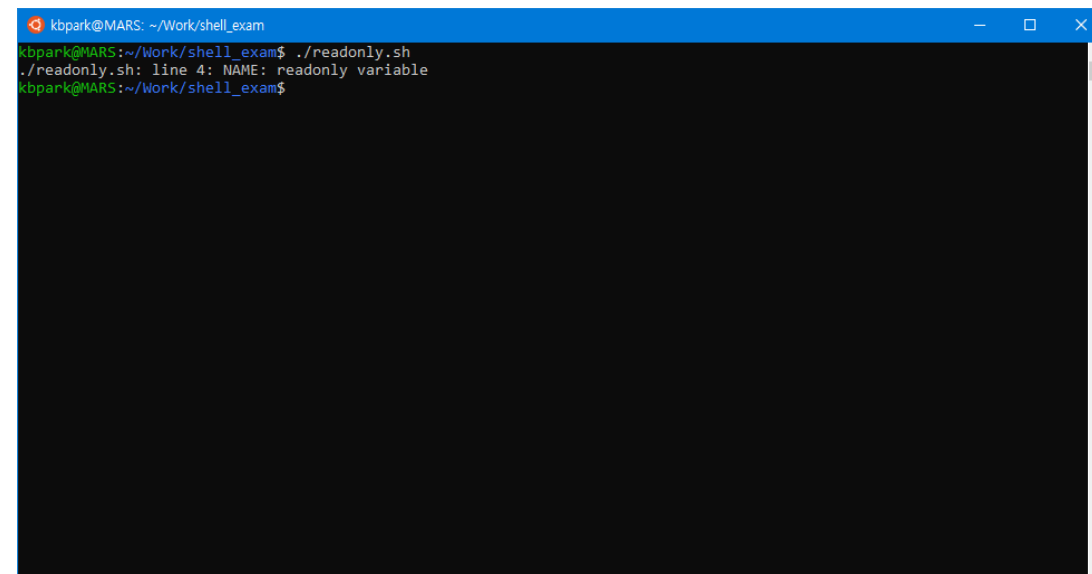


```
kbpark@MARS: ~/Work/shell_exam
kbpark@MARS:~/Work/shell_exam$ ./readonly.sh
./readonly.sh: line 4: NAME: readonly variable
kbpark@MARS:~/Work/shell_exam$
```

LINUX - SHELL SCRIPT

Unsetting Variables

- 변수에 할당된 값을 해제하면 더 이상 변수에 접근할 수 없다.
 - `#!/bin/bash`
 - `NAME="Lucas"`
 - `unset NAME`
 - `echo $NAME`
- 위의 변수를 실행하게 되면 어떠한 것도 출력되지 않으며, 읽기 전용으로 선언된 변수는 `unset` 할 수 없다.



```
kbpark@MARS: ~/Work/shell_exam
kbpark@MARS:~/Work/shell_exam$ ./readonly.sh
./readonly.sh: line 4: NAME: readonly variable
kbpark@MARS:~/Work/shell_exam$
```

LINUX - SHELL SCRIPT

Variable Types

- 셸이 실행하기 위해서는 다음과 같은 3가지의 주요 변수가 존재한다.
 - 지역변수
 - 셸의 인스턴스에 존재하는 변수로 기본적으로 셸에서 지정한 모든 변수는 전역 변수로 선언되기 때문에 지역변수를 사용하기 위해서는 local 키워드를 반드시 사용해야 한다.
 - 혹은 셸을 실행할 때 인자 값으로 넘겨줄 수 있다.
 - 환경변수
 - 셸 스크립트를 통해 작성된 프로그램 중에서 정상적으로 동작하기 위한 변수이다.
 - 선언된 변수는 모든 자식 프로세스에서 접근하여 사용할 수 있으며, 프로그램이 실행하기 위해 참조되는 경우 외에는 사용하면 안 된다.

LINUX - SHELL SCRIPT

Variable Types

■ 셸 변수

- 셸이 동작하기 위해 필요한 특수 변수이다.
- set 명령을 통해 셸 변수를 확인할 수 있다.
- 우리가 흔히 알고 있는 셸 변수로는 \$PATH, \$HOME 등이 있다.

```
$ set
'!'=0
'#'=0
'$'=985
'*'=( )
-=569JNRXZghiklms
0=-zsh
'?'=0
@=( )
ARGC=0
BG
CDPATH=''
COLORFGBG='7;0'
COLORTERM=truecolor
COLUMNS=114
COMMAND_MODE=unix2003
COMP_WORDBREAKS=:
CPUTYPE=x86_64
CURRENT_BG=NONE
```

LINUX - SHELL SCRIPT - SHELL 특수 변수

예약 변수 (Reserved Variable)

- 일반적인 프로그래밍 언어에서 사용하는 예약 변수와 동일한 기능을 담당한다고 생각하면 된다. 쉘 프로그래밍을 작성할 때, 예약 변수를 사용하면 보편적인 실행환경으로 작성할 수 있으므로 편리하게 사용할 수 있다.

LINUX - SHELL SCRIPT - SHELL 특수 변수

예약 변수 (Reserved Variable)

변수명	설명
HOME	사용자 홈 디렉터리
PATH	실행 파일을 찾을 경로
LANG	프로그램 사용시 기본 지원되는 언어
FUNCNAME	현재 함수 이름
SECONDS	스크립트가 실행된 시간 (초 단위)
SHLVL	중첩된 셸 레벨 (현재 실행중인 셸 레벨 수이며, Linux 배포판에 따라 다름)
SHELL	로그인해서 사용하고 있는 셸
PPID	부모 프로세스의 PID
BASH	BASH 실행 파일 경로
BASH_ENV	스크립트 실행시 BASH 시작 파일을 읽을 위치 변수

변수명	설명
BASH_VERSION	현재 설치된 BASH 버전
BASH_VERSINFO	배열로 상세 정보 출력
MAIL	메일 보관 경로
MAILCHECK	메일 확인 시간
OSTYPE	운영체제 종류
TERM	터미널의 종류
HOSTNAME	호스트 이름
HOSTTYPE	시스템 하드웨어 종류
MACHTYPE	머신 종류
LOGNAME	로그인 이름

LINUX - SHELL SCRIPT - SHELL 특수 변수

예약 변수 (Reserved Variable)

변수명	설명
UID	사용자 UID
TMOUNT	로그아웃할 시간, 0일 경우 무제한
USER	사용자의 이름
USERNAME	사용자 이름
GROUPS	사용자 그룹 (/etc/passwd)
HISTFILE	히스토리 파일 경로
HISTFILESIZE	히스토리 파일 사이즈
HISTSIZE	히스토리에 저장된 개수
HISTCONTROL	중복되는 명령에 대한 기록 유무
DISPLAY	X 디스플레이 이름
IFS	입력 필드 구분자

변수명	설명
VISAUL	VISUAL 편집기 이름
EDITOR	기본 편집기 이름
COLUMNS	터미널의 컬럼 수
LINES	터미널의 라인 수
LS_COLORS	ls 명령의 색상 관련 옵션
PS1	기본 프롬프트 스트링. 기본값은 <code>`\s-\v\$ '</code>
PS2	긴 문자 입력을 위해 나타나는 문자열. 기본 값은 <code>></code>
PS3	셸 스크립트에서 select 사용시 프롬프트 변수(기본 값: <code>#?</code>)
PS4	셸 스크립트 디버깅 모드의 프롬프트 변수