Clickseo.com

GNU/Linux

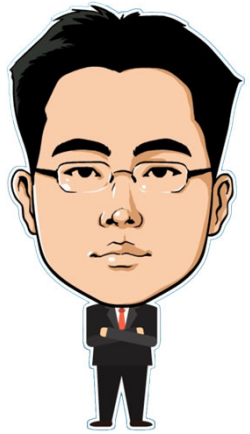
쉘 프로그래밍

(Shell Programming)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

- 유닉스 쉘
- GNU Bash
- 쉘 프로그래밍

유닉스 쉘

- 유닉스 쉘
  - Bourne Shell 계열
  - C Shell 계열
  - 쉘 정보
- GNU Bash
- 쉘 프로그래밍

유닉스 쉘 (1/2)

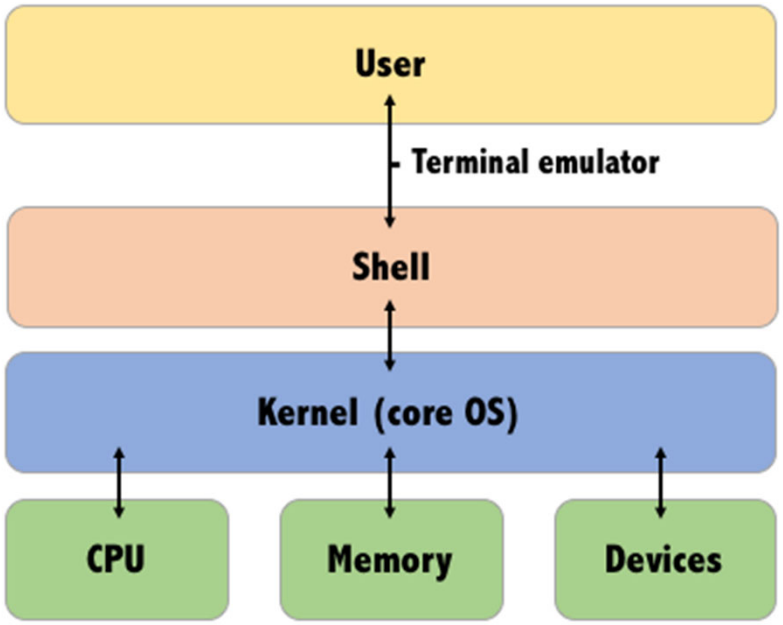
쉘(Shell)

운영체제 기능과 서비스 구현을 위해 인터페이스 제공하는 프로그램

- 명령어를 실행시키는 명령어 해석기
  - 사용자의 명령어를 입력 받아 기계어의 형태로 변환하여 커널에 전달하는 인터페이스 역할

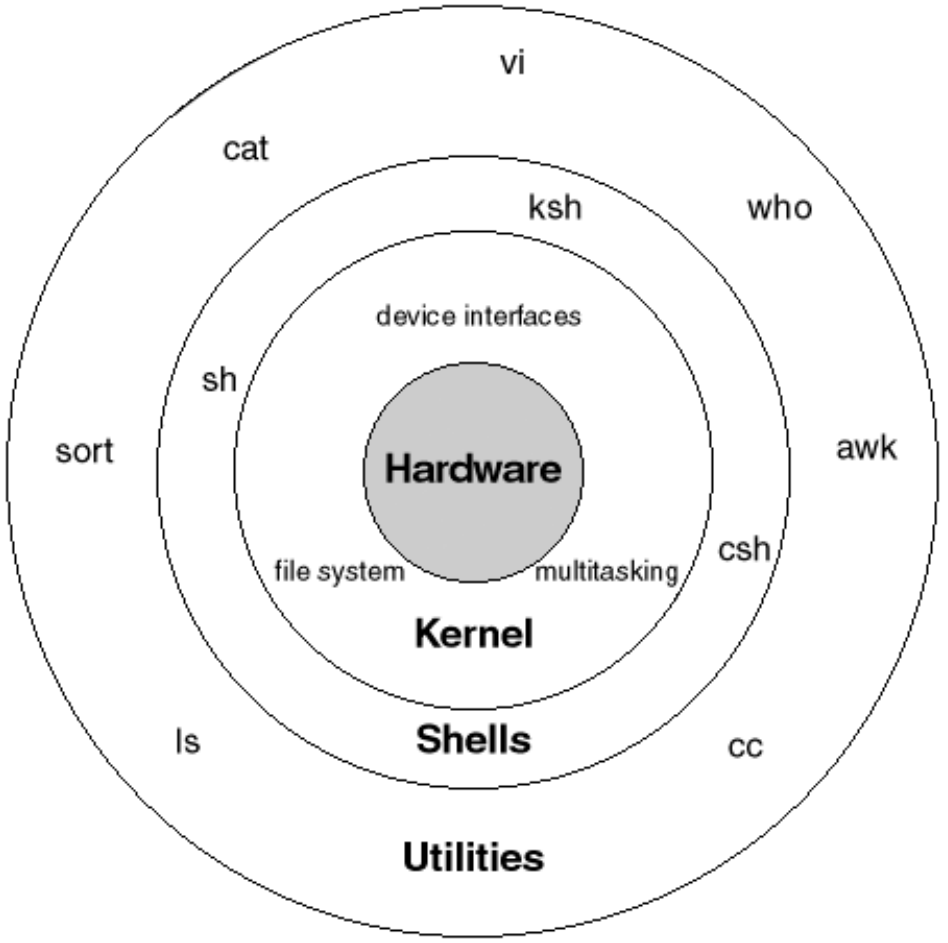
쉘의 종류

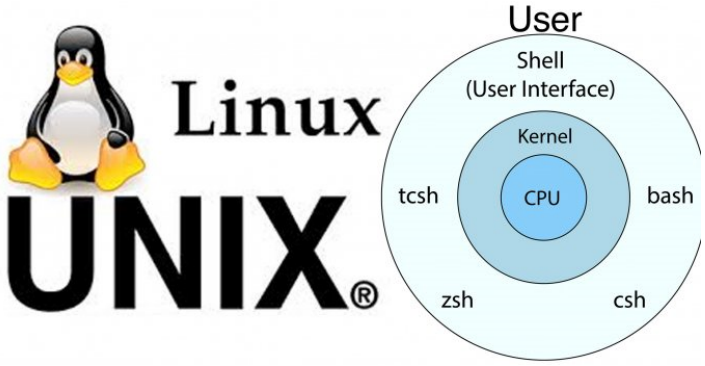
- 텍스트 기반 CLI 쉘
  - 유닉스 쉘
  - 윈도우 명령 프롬프트
- 그래픽 기반 GUI 쉘
  - 윈도우 탐색기
  - 윈도우 PowerShell
  - 맥 OS 파인더(Finder)



유닉스 쉘 (2/2)

- **UNIX Shell**
  - Bourne Shell 계열
  - C Shell 계열



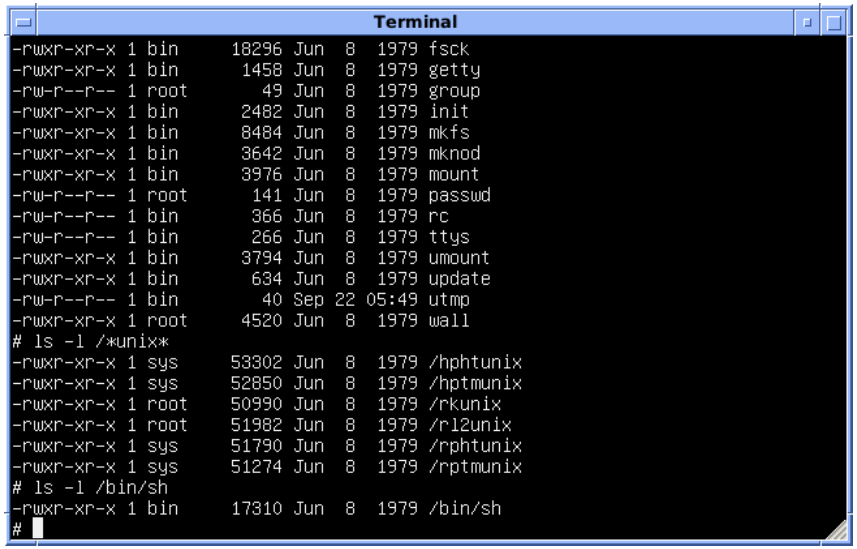


유닉스 쉘

Bourne Shell, C Shell 계열

Bourne Shell 계열 (1/6)

- **Bourne Shell : sh**
  - **1977년, AT&T Bell 연구소의 개발자 Steven Bourne**
    - 가장 오랫동안 UNIX 시스템의 표준 구성 요소
    - UNIX Version 7 의 기본 쉘로 **Thompson Shell**을 대체
    - 대화형 인터프리터로 사용
    - 제어흐름과 변수를 포함한 프로그래밍이 가능한 스크립트 언어로도 사용
  - **기본 Login Shell**
    - **SunOS-5.x, FreeBSD**
  - **라이선스 : CDDL**



*** Thompson Shell**

1971년, 켄 톰슨(Kenneth Thompson)이 개발하여 최초 유닉스 버전에 도입된 최초의 유닉스 쉘

[출처 : "Bourne shell", WIKIPEDIA]

Bourne Shell 계열 (2/6)

- **Korn Shell : ksh -- kornshell.org**
  - **1983년, AT&T Bell 연구소의 David Korn 개발**
    - C Shell과 달리 Bourne Shell과 하위 호환성을 유지한다.
    - C Shell 의 수 많은 기능(특징)을 포함하면서 처리 속도가 빠르다는 장점이 있다.
      - 명령어 완성 기능
      - Alias 기능
      - 히스토리 기능
  - **기본 Login Shell**
    - **AIX, HP-UX**
  - **라이선스 : EPL(Eclipse Public License)**

Bourne Shell 계열 (3/6)

- **GNU Bash : gnu.org/software/bash/**
  - **1989년 06월, Brian Fox에 의해서 개발**
    - **최종 목적 : IEEE POSIX Shell과 도구명세에 호환되도록 하는 것**
      - Bourne Shell의 기능을 추가 발전 시키면서, Korn Shell과 C Shell의 유용한 특징들도 지원
      - 쉘 프로그래밍 언어에 있어서는 Bourne Shell 과 호환
    - **Linux 배포판과 macOS 에서 기본 쉘**
    - **윈도우 10 에서 개발자들을 위한 도구로써 Bash 지원**
  - GNU Bash 릴리즈
    - 2009년 04월, Bash 4.0
    - **2019년 01월, Bash 5.0**
  - 라이선스 : GNU GPL License
  - 프로그래밍 언어 : C

Bash Git : savannah.gnu.org

https://git.savannah.gnu.org/cgit/bash.git/

Bourne Shell 계열 (4/6)

- **Z Shell : zsh -- zsh.org**
  - **1990년, 폴 팔스타드(Paul Falstad)가 개발한 확장형 Bourne Shell**
    - 강력한 히스토리 기능, 향상된 명령행 편집 기능
    - 파일명 중간에서부터 자동 완성 기능 가능, 탭이나 화살표 키를 이용해 선택 가능
  - 라이선스 : MIT License 유형
  - 프로그래밍 언어 : C

[출처 : "Z shell", WIKIPEDIA]

zsh : sourceforge.net

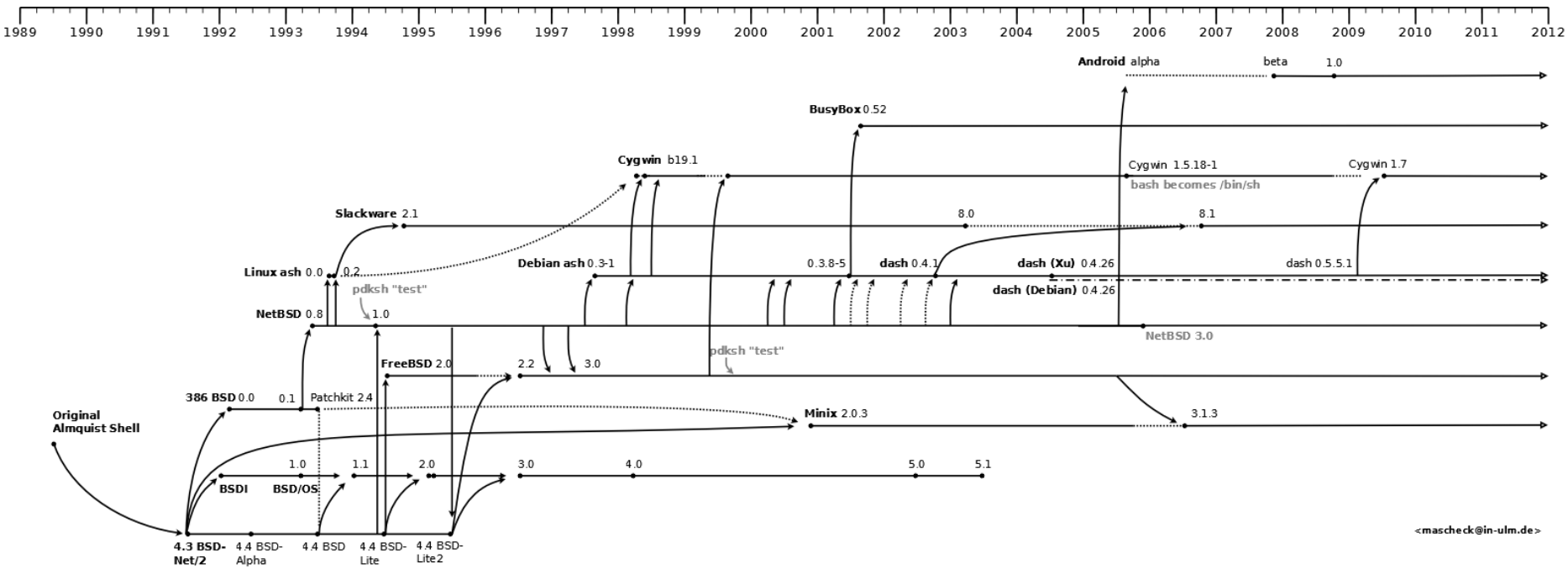
https://sourceforge.net/projects/zsh/

Bourne Shell 계열 (5/6)

- **A Shell**(Almquist shell) : **ash**
  - 1989년, 경량 유닉스 쉘
  - 1990 년대 초반 BSD 버전 유닉스에서 기존 Bourne 쉘을 대체
    - ash 파생판은 **FreeBSD, NetBSD, DragonFly BSD, Minix, Android, 일부 리눅스 배포판**에 **기본 쉘(/bin/sh)**로 설치된다.
  - 라이선스 : BSD License 유형
  - **Debian A Shell**(Debian Almquist Shell) : **dash**
    - 1997년, 최초 버전 발표 이후 2002년에 **dash** 로 이름 변경
    - **Debian/Ubuntu Linux 및 POSIX 준수**

Bourne Shell 계열 (6/6)

- **A Shell**(Almquist shell) : 다양한 변형 판
  - 다양한 A Shell 변형 판



[출처 : "Ash (Almquist Shell) Variants", https://www.in-ulm.de/~mascheck/]

C Shell 계열 (1/2)

- **C Shell : csh**
  - **1978년, Bill Joy가 버클리 버전 유닉스의 쉘로 개발**
    - C 언어와 유사하며, 강력한 프로그램 작성 기능, 대화식 방식 사용자 환경
  - **라이선스 : BSD License**
  - **프로그래밍 언어 : C**

csh OpenGrok : http://bxr.su/NetBSD/

http://bxr.su/NetBSD/bin/csh/

C Shell 계열 (2/2)

- **TC Shell : tcsh -- tcsh.org**
  - **1982년, C Shell 기반의 호환 가능한 유닉스 쉘**
    - 프로그래밍 가능한 명령 줄 완성
    - 명령 줄 편집 및 기타 몇 가지 기능을 갖춘 C Shell
  - **기본 로긴 쉘**
    - **초기 버전의 macOS X**
  - **라이선스 : BSD License**
  - **프로그래밍 언어 : C**

tcsh Git : github.com

https://github.com/tcsh-org/tcsh

유닉스 쉘

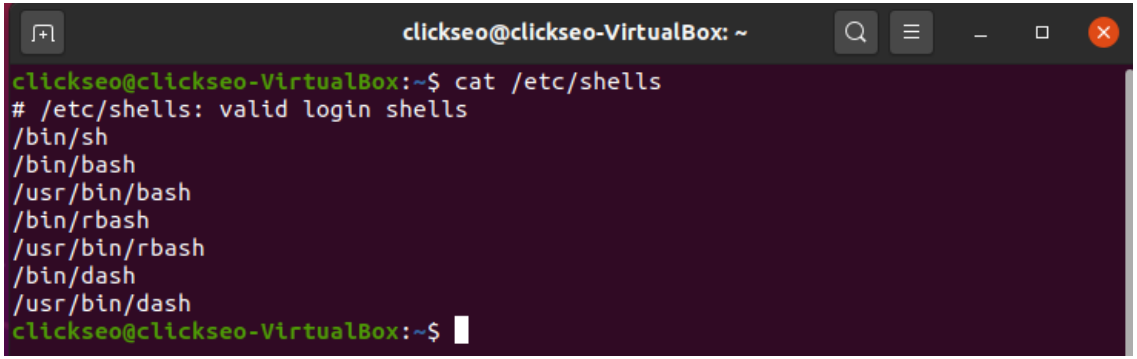
쉘 정보

쉘 정보 (1/3)

● 쉘 확인

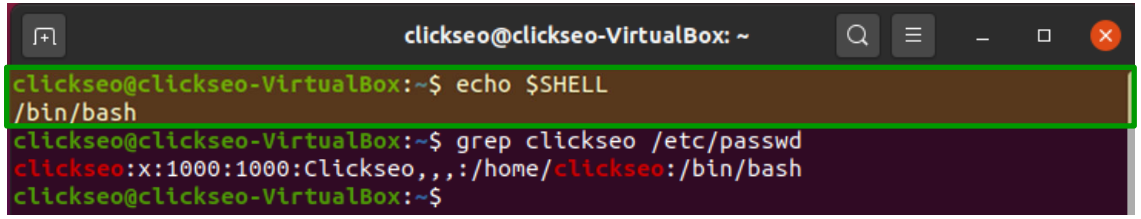
❍ 시스템에서 사용 가능한 쉘 확인 : /etc/shells

```
[clickseo@localhost ~]$ cat /etc/shells
```



❍ 현재 쉘 확인

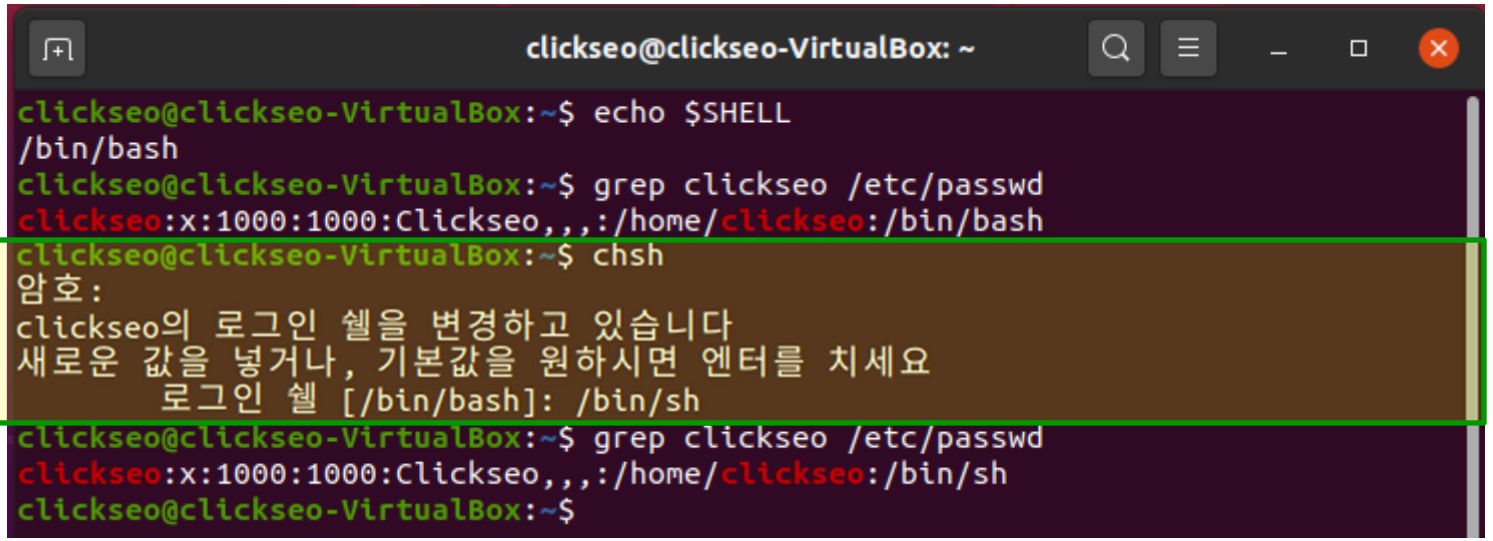
```
[clickseo@localhost ~]$ echo $SHELL
```



쉘 정보 (2/3)

- **쉘 변경**
  - 리눅스 쉘 변경 : chsh

쉘 변경
/bin/sh

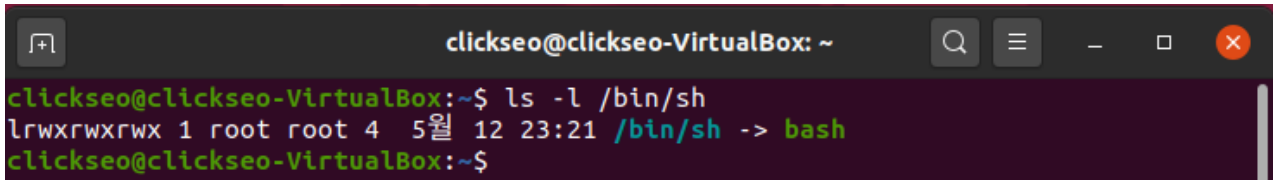


logout 하고 다시 login 시 적용 된다.

쉘 정보 (3/3)

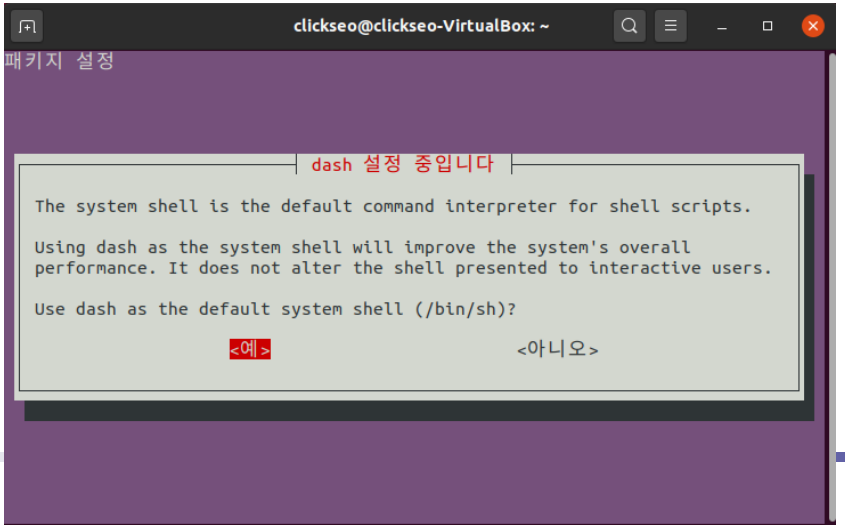
● 쉘 변경 : Ubuntu Linux

❍ Ubuntu Linux 기본 쉘 변경



- **dpkg-reconfigure** : 설치된 패키지 설정 변경

```
[clickseo@localhost  ~]$ sudo dpkg-reconfigure dash
```



GNU Bash

- 유닉스 쉘
- **GNU Bash**
  - 환경 설정 파일
  - 내부 명령어
  - 쉘 활용
- 쉘 프로그래밍

GNU Bash (1/3)

- **GNU Bash** : gnu.org/software/bash/
  - 1989년 06월, Brian Fox에 의해서 개발
    - 최종 목적 : IEEE POSIX Shell과 도구명세에 호환되도록 하는 것
      - Bourne Shell의 기능을 추가 발전 시키면서, Korn Shell과 C Shell의 유용한 특징들도 지원
      - 쉘 프로그래밍 언어에 있어서는 Bourne Shell 과 호환
    - Linux 배포판과 macOS 에서 기본 쉘
    - 윈도우 10 에서 개발자들을 위한 도구로써 Bash 지원
  - GNU Bash 릴리즈
    - 2009년 04월, Bash 4.0
    - 2019년 01월, Bash 5.0
  - 라이선스 : GNU GPL License
  - 프로그래밍 언어 : C

Bash Git : savannah.gnu.org

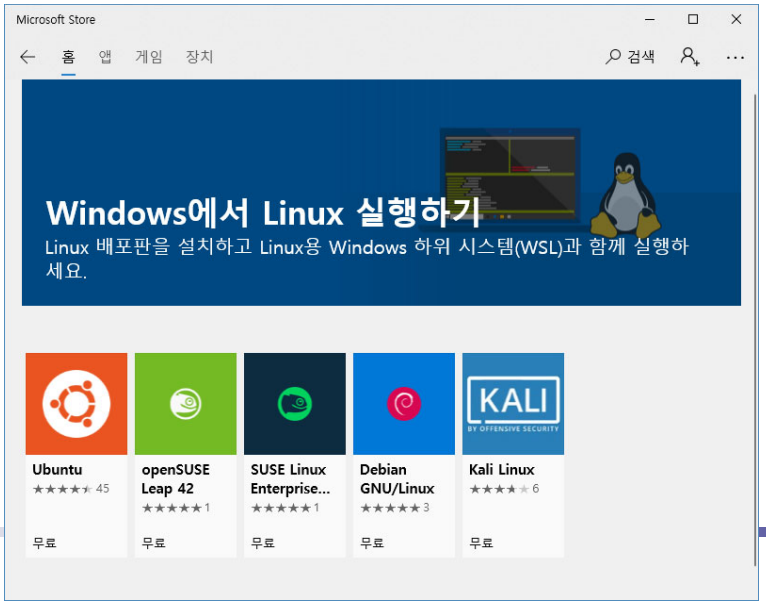
https://git.savannah.gnu.org/cgit/bash.git/

GNU Bash (2/3)

● 리눅스를 위한 윈도우 서브 시스템

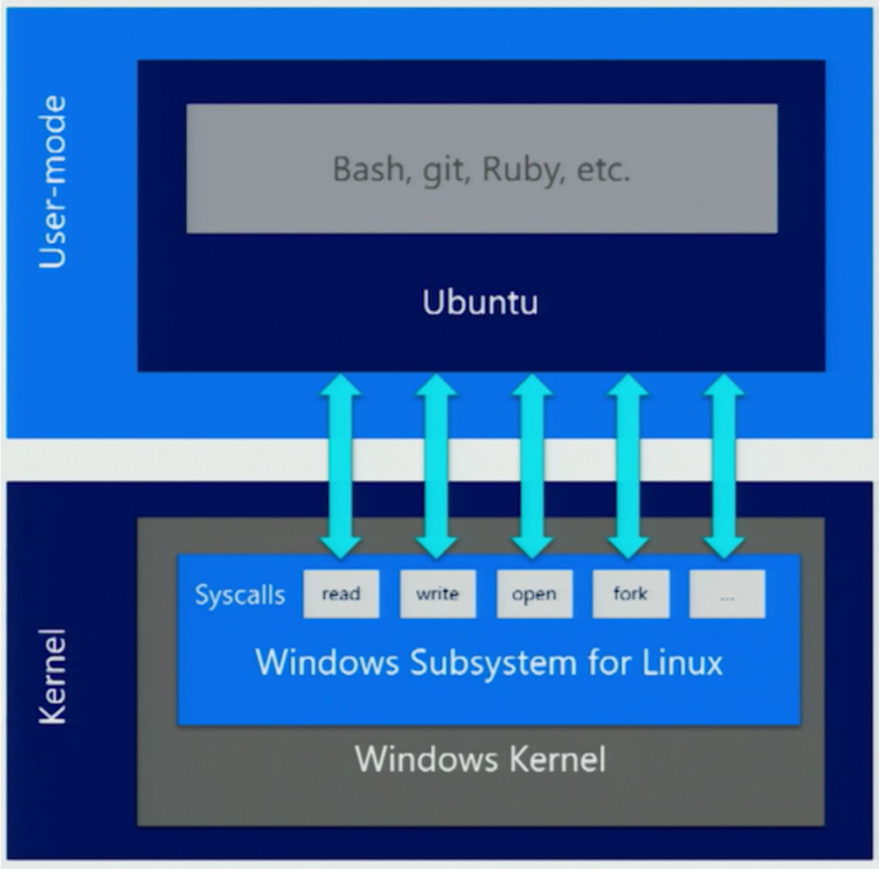
❍ WSL(Windows Subsystem for Linux)

- **윈도우 10** 및 **윈도우 서버 2019** 에서 기본적으로 리눅스 바이너리 실행 파일을 실행하기 위한 계층으로 **호환 커널 인터페이스를 제공**한다.
  - **Windows Services for UNIX**를 대체
- **개발자 모드 및 윈도우 기능 사용 설정**
  - 1단계 : Windows 설정 > 업데이트 및 보안 > 개발자용 > 개발자 모드
  - 2단계 : 제어판 > 프로그램 > Windows 기능 켜기/끄기 > Windows Subsystem for Linux



GNU Bash (3/3)

- **리눅스를 위한 윈도우 서브 시스템 : WSL 리눅스 실행**
  - WSL를 사용하여 윈도우에서 리눅스 실행

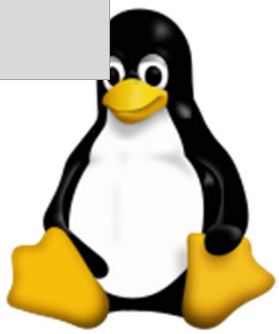


[출처 : "윈도우 10에서 Bash shell 지원", developer.microsoft.com]

GNU Bash

환경 설정 파일

내부 명령어, 쉘 활용



GNU Bash : 환경 설정 파일 (1/2)

● 환경 설정 파일

❍ login shell

- **/etc/profile**
  - 시스템 전반적인 환경 설정, 사용자 로그인 시 시스템 초기화
- **~/.bash_profile --> ~/.bash_login --> ~/.profile**
  - **로그인할 때 읽어 들이는 설정파일로 사용자별 환경설정**
  - 해당 파일 존재 시 순서대로 호출된다.
  - 주요 설정 내용 : 검색경로, 터미널 종류, 환경변수 등
- **~/.bashrc**
  - **새로운 쉘이 실행될 때마다 실행되며, 쉘을 위한 쉘 스크립트로 서브 쉘**

- 각종 설정 파일들은 새로운 사용자를 등록하면, **/etc/skel** 디렉터리에 기본값으로 저장되어 있는 파일들을 홈 디렉터리에 복사하여 생겨나는 것이다.

❍ logout

- **~/.bash_logout**
  - **로그인 쉘이 종료되면서 실행된다.**

GNU Bash : 환경 설정 파일 (2/2)

● 기타 환경 설정 파일

❍ /etc/profile.d 디렉터리

- 터미널을 사용할 때 파일 및 디렉터리에 대한 색상 설정
- 사용자 언어 설정

❍ /etc/skel 디렉터리

- 슈퍼유저인 root 가 새로운 사용자 생성 시, 사용자별 홈 디렉터리에 기본 값으로 복사될 파일이 존재하는 디렉터리

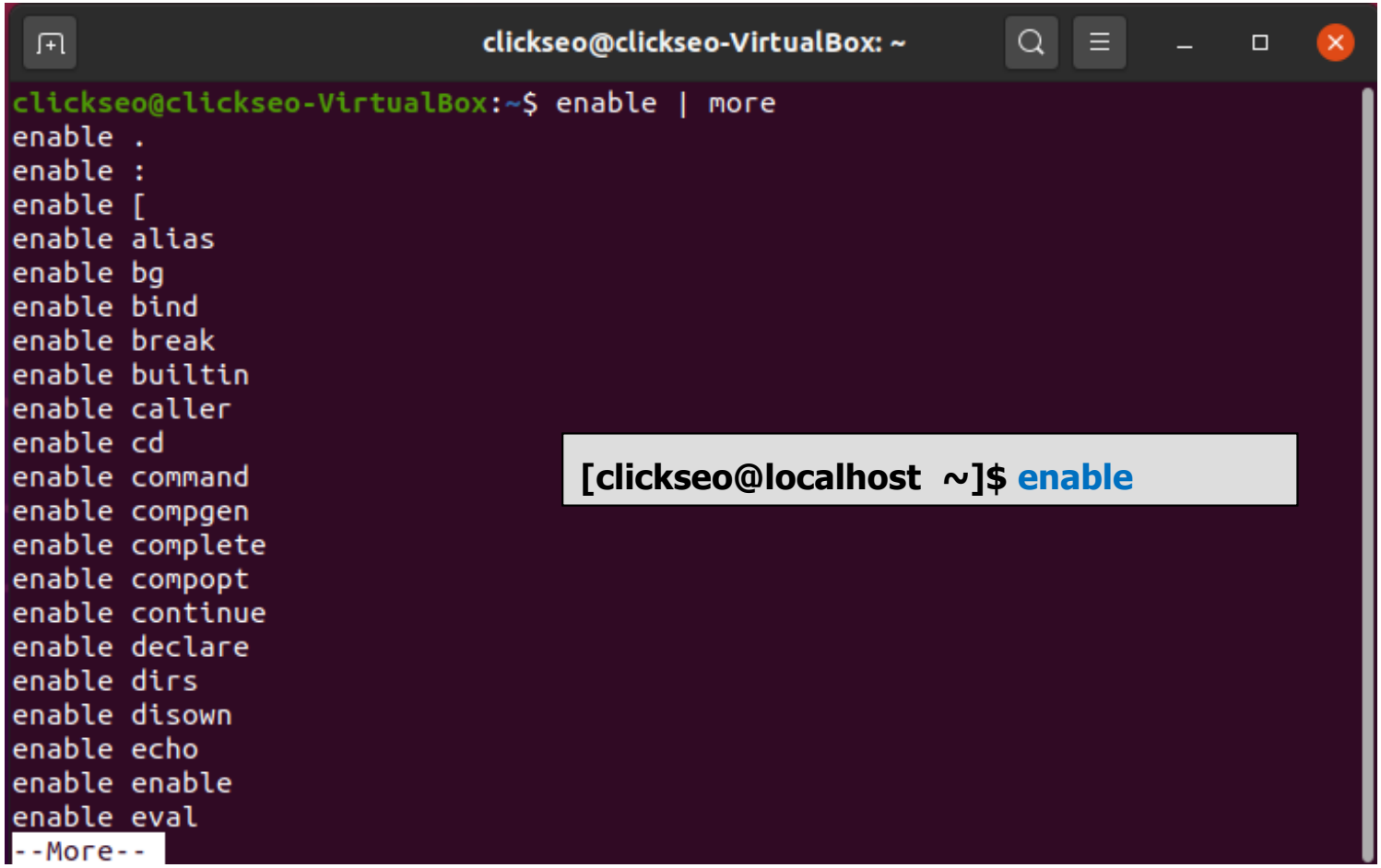
GNU Bash : 내부 명령어 (1/3)

- **쉘 내부 명령어(Built-in Command)**
  - 쉘 프로그램 자체적으로 처리하는 명령어

내부 명령어	내용
cd	디렉터리 변경
pwd	현재 작업 디렉터리의 절대 경로 출력
history	이전에 작업한 명령어 리스트 출력
exit	종료
echo	문자열 출력
read	사용자로부터 값을 읽어 들인다.
alias	명령어에 대한 별칭(alias) 지정
export	환경 변수 설정
set	변수 설정
unset	변수 초기화

GNU Bash : 내부 명령어 (2/3)

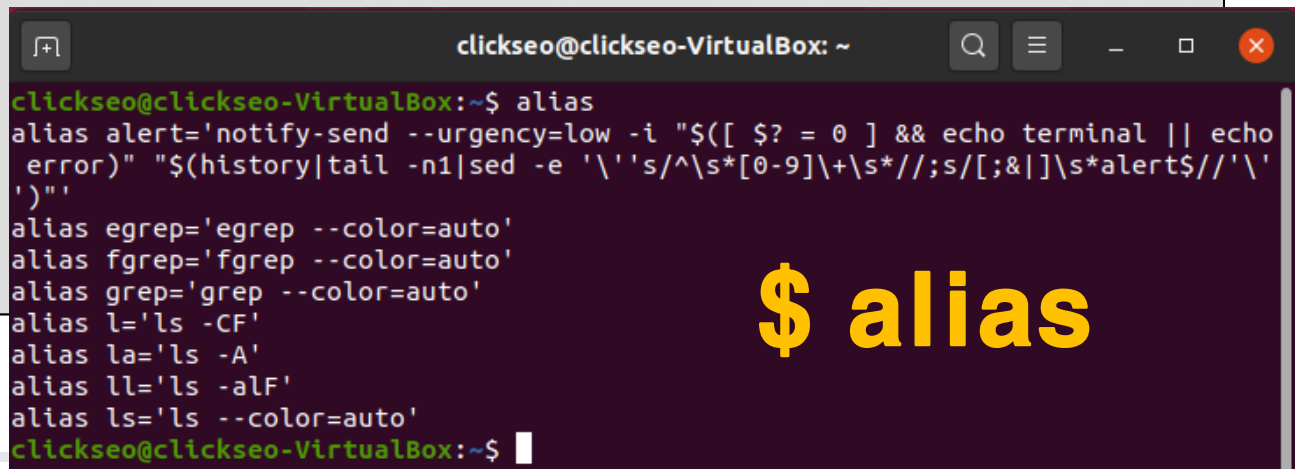
- **쉘 내부 명령어** : enable



GNU Bash : 내부 명령어 (3/3)

- **쉘 내부 명령어** : alias
  - 기존 명령어와 옵션의 조합에 대하여 별칭을 지정한다.

```
[clickseo@localhost  ~]$ vi .bashrc
...
        alias ls='ls --color=auto'
        alias grep='grep --color=auto'
        alias fgrep='fgrep --color=auto'
        alias egrep='egrep --color=auto'
...
alias ll='ls -alF'
alias la='ls –A'
alias l='ls –CF'
...
```



GNU Bash : 쉘 활용 (1/3)

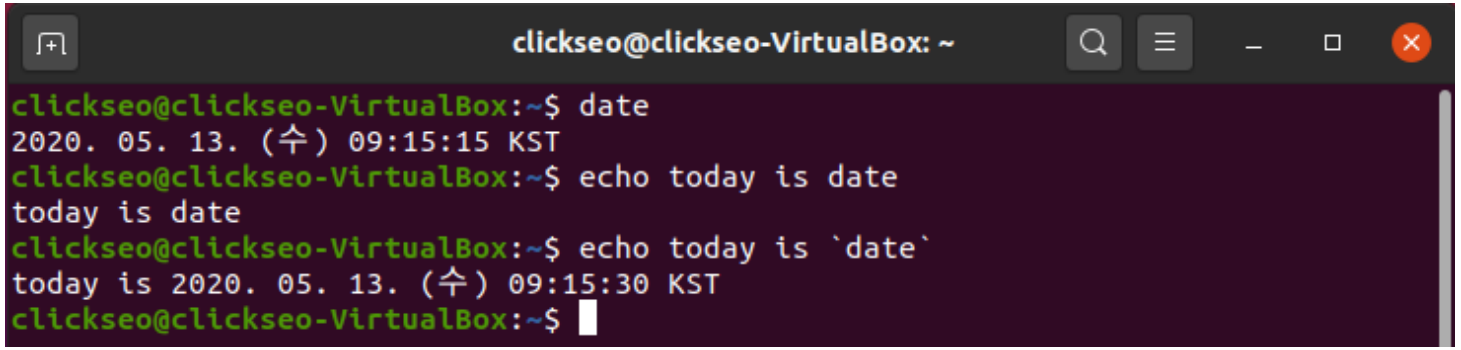
● 메타 문자와 역 따옴표

❍ 명령어에 메타 문자 사용

- * : 길이에 관계없이 모든 문자를 가리킨다.
- ? : 정하지 않은 하나의 문자만을 가리킨다.

❍ 명령어에 역 따옴표(backquote) : `

```
[clickseo@localhost  ~]$ date
[clickseo@localhost  ~]$ echo today is date
[clickseo@localhost  ~]$ echo today is`date`
```



GNU Bash : 쉘 활용 (2/3)

Redirection과 Pipeline

Redirection : > , >>

```
[clickseo@localhost  ~]$ cat > test.text
```

- 파일에 새로운 내용을 저장한다.

```
[clickseo@localhost  ~]$ cat >> test.text
```

- 이미 작성되어 있는 파일 내용 뒤에 첨부하는 형식으로 저장된다.

Pipeline : |

- [명령 1] | [명령 2] --> 명령 1의 결과가 명령 2의 입력으로 보내져 처리된다.

GNU Bash : 쉘 활용 (3/3)

● 무조건부와 조건부 실행

❍ 무조건부 실행 : 쌍반점(Semicolon) --> ;

- 앞 명령의 성공 여부와 상관없이 무조건적으로 실행된다.
- [명령 1]; [명령 2] --> **명령 1이 실행된 후에 명령 2가 실행된다.**

```
[clickseo@localhost  ~]$ date > datae.txt; ls; cat date.txt
```

❍ 조건부 실행

- **&&** : 이전 명령에 성공하면 다음 명령을 실행
- **||** : 이전 명령에 실패할 경우에 다음 명령을 실행

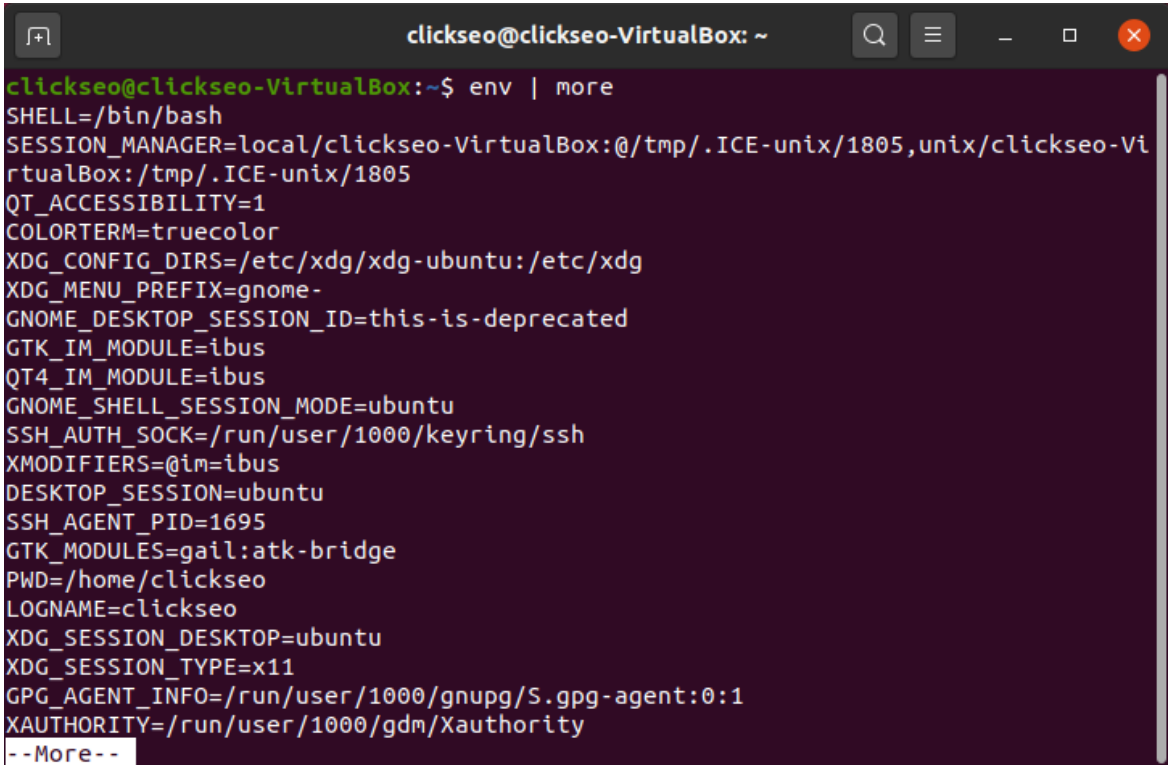
쉘 프로그래밍

- UNIX 쉘
- GNU Bash
- **쉘 프로그래밍**
  - 환경 변수
  - 내장 변수
  - 쉘 프로그래밍 기초

환경 변수 (1/5)

- **쉘 환경 변수**
  - 현재 설정 된 환경 변수 값을 모두 확인

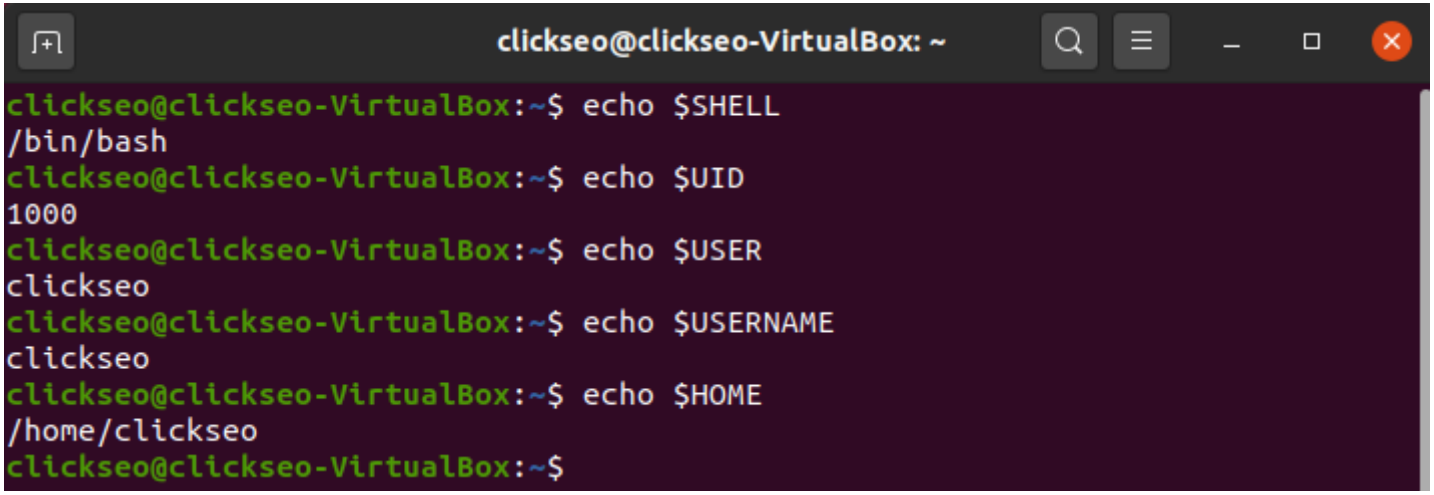
```
[clickseo@localhost  ~]$ env
```



환경 변수 (2/5)

- **쉘 환경 변수** : 설정 값
  - 개별적으로 환경 변수 설정 값을 확인

```
[clickseo@localhost  ~]$ echo $변수명
```



환경 변수 (3/5)

- **쉘 환경 변수**

환경변수	설 명
$HOME	사용자의 홈 디렉토리를 설정
$PATH	실행 프로그램의 탐색 경로
$SHELL	현재 쉘의 경로와 이름
$BASH	사용하고 있는 Bash의 쉘 경로
$BASH_VERSION	사용하고 있는 Bash의 버전
$HOSTNAME	현재 컴퓨터의 이름
$LS_COLORS	ls 명령을 사용할 때 파일의 종류마다 나타나는 색의 결정
$COLUMNS	터미널의 행수
$ENV	환경 지정 파일의 위치
$HISTFILE	히스토리 파일의 경로
$HISTSIZE	히스토리의 개수

환경 변수 (4/5)

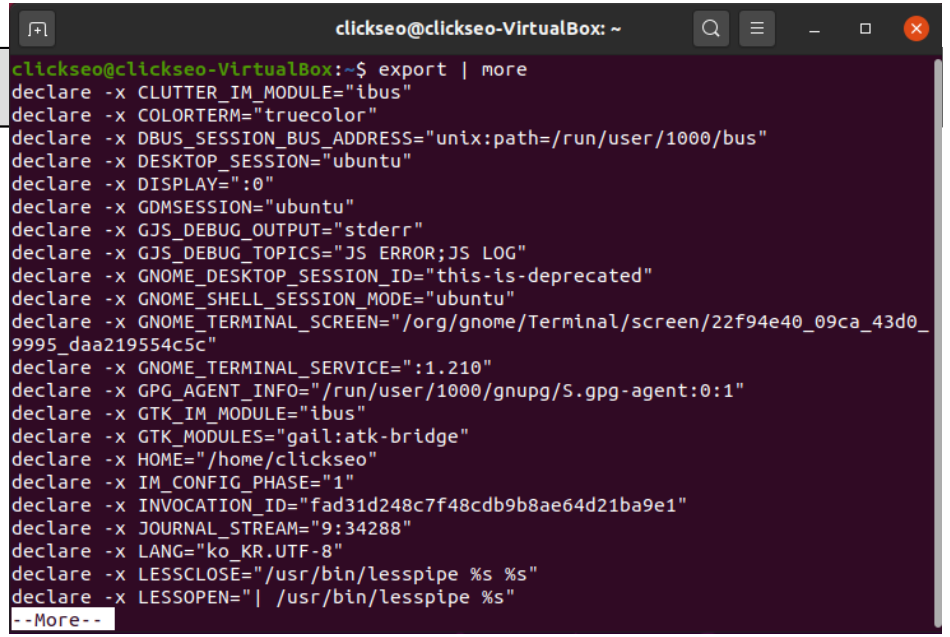
- **쉘 환경 변수**

환경변수	설 명
$LINES	터미널의 라인 수
$MANPATH	도움말이 있는 경로
$PWD	현재 위치(절대경로)
$UID	사용자 UID
$USER	사용자
$USERNAME	사용자 이름
$PS1	검색경로, 터미널 종류, 환경변수 등을 설정하고, 그 외 로그인 시점에 실행시키고 싶은 명령, 시스템에 대한 정보를 보여주는 명령 등을 수행

환경 변수 (5/5)

- **쉘 환경 변수** : 설정 값 변경

```
$ export [환경변수명]=[변수 값]
```



- 사용 예

```
[clickseo@localhost  ~]$ export HISTSIZE=500
```

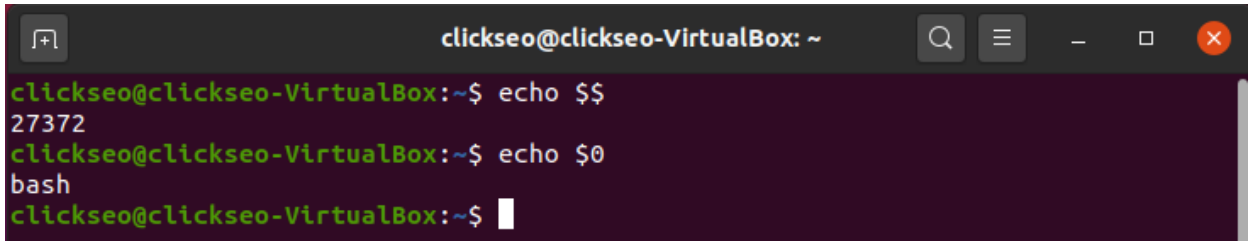
  - HISTSIZE 환경 변수 1000 --> 500으로 변경

~/.bash_profile 내의 변수 값 설정/변경

내장 변수

- **쉘 내장 변수**

이 름	내 용
$$	실행중인 프로세스의 PID
$0	명령어 실행 시 명령어를 저장하는 변수
$?	마지막에 실행한 명령어의 종료 값
$!	마지막 자식 프로세스의 PID
$_	마지막 명령어나 명령어에 전달된 인수 값
$#	명령어에 전달된 매개변수의 개수
$1 ... $n	명령어 실행 시 매개 변수들을 저장하는 변수
$*	명령어 실행 시 전달된 매개변수 전체를 하나의 문자열로 저장하는 변수



쉘 프로그래밍 기초

쉘 변수와 입출력

쉘 변수와 입출력 (1/3)

● 쉘 변수

❍ 쉘 변수는 자료유형이 없다.

- **즉, 아무 값이나 다 넣을 수 있다.**
- **쉘 변수는 기본적으로 데이터를 문자열로 저장한다.**
  - 수치를 대입해도 실제 수치가 아닌 문자열이 저장된다.
  - 계산이 필요할 경우는 자동으로 수치로 변환하여 계산 후 다시 문자열로 저장된다.

❍ 쉘 변수는 처음 사용될 때 만들어진다.

- **즉, 미리 선언할 필요가 없다.**
- 쉘 변수는 유닉스 명령과 마찬가지로 대소문자에 구별이 있다**(대소문자 구별).**
- 첫 번째 문자는 문자로 시작한다.
- 시스템 변수와 내부 및 외부 명령어는 쉘 변수로 사용할 수 없다.

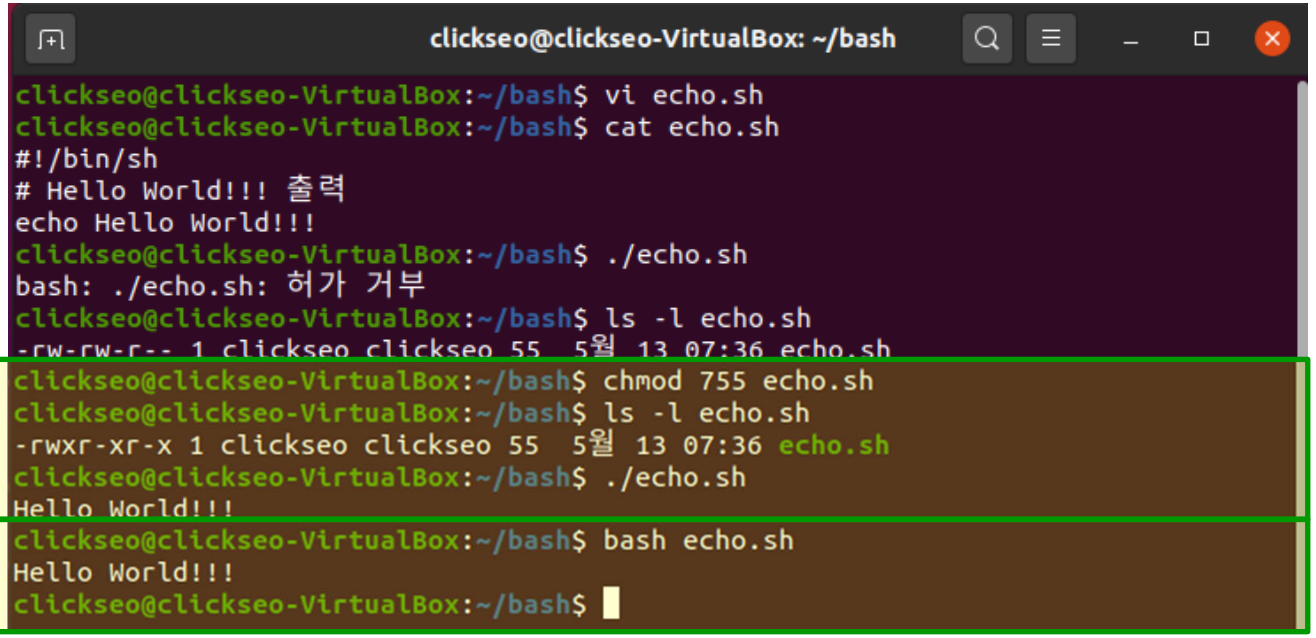
- 쉘 변수의 값을 사용할 때는 변수명 앞에 **"$"** 를 붙여서 사용한다.
- 쉘 변수에 값을 대입할 때는 **"$"**를 사용하지 않는다.

쉘 변수와 입출력 (2/3)

● 쉘 입출력

❍ 사용자 출력 : echo

```
#!/bin/bash
# Hello World!!! 출력
echo Hello World!!!                    # echo "Hello World!!!"
```



접근 권한 변경 (실행권한 부여)

bash 명령으로 실행

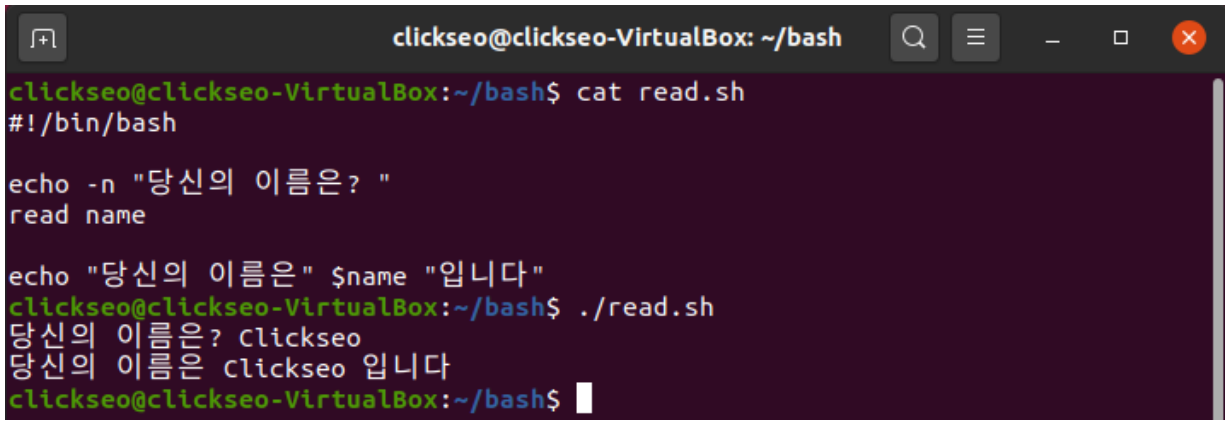
쉘 변수와 입출력 (3/3)

● 쉘 입출력

❍ 사용자 입력 : read

- C 쉘 에서는 $< 을 사용해 사용자의 입력을 받았다.
- **Bourne Shell 에서는 read 명령어를 사용한다.**

```
#!/bin/bash
echo -n "당신의 이름은?"
read name
echo "당신의 이름은" $name "입니다."
```



쉘 프로그래밍 기초

제어흐름

제어흐름 (1/8)

● 조건문

❍ 산술 비교 연산자

비교 연산자	설 명
A -eq B	양변이 **같은지** 검사(==)
A -ne B	양변이 **다른지** 검사(!=)
A -gt B	A가 B보다 **큰지** 검사(>)
A -lt B	A가 B보다 **작은지** 검사(<)
A -ge B	A가 B보다 **크거나 같은지** 검사(>=)
A -le B	A가 B보다 **작거나 같은지** 검사(<=)

제어흐름 (2/8)

- **선택 구조** : 이중 선택
  - if ~ then

```
if  조건  then
        명령문
fi
```

```
#!/bin/bash
man=10
woman=20
if [ $man -lt $woman ]
then
        echo woman
fi
```

제어흐름 (3/8)

- **선택 구조** : 이중 선택
  - if ~ then ~ else

```
#!/bin/bash
man=10
woman=20
if [ $man -lt $woman ]
then
        echo woman
else
        echo man
fi
```

제어흐름 (4/8)

- **선택 구조** : 다중 선택
  - if ~ then ~ elif ~ else

```
#!/bin/bash
man=10
woman=20
if [ $man -lt $woman ]
then
        echo woman
elif [ $man -eq $woman ]
then
        echo same
else
        echo man
fi
```

제어흐름 (5/8)

선택 구조 : 다중 선택

case

- ;; : C에서의 **break**와 의미가 같다.

```
case $변수 in
        패턴1:
        명령문1;;

        ...

        패턴n:
        명령문n;;
esac
```

제어흐름 (6/8)

● 반복 구조 : for

❍ for 문

- 지정된 변수는 **'in'** 뒤에 나오는 문자를 순서대로 한 단어 씩 받고, **'do'** 이후에 나오는 명령문을 입력 받은 문자가 없을 때까지 반복 수행한다.

```
for 변수 in list1 list2 list3 …
do
        명령문
done
```

```
#!/bin/bash
for i in C l i c k s e o
do
        echo $i
done
```

제어흐름 (7/8)

● 반목 구조 : while

❍ while 문

- 해당 조건을 만족하는 동안 **do** 이후에 나오는 명령문을 반복 수행한다.

```
while 조건문
do
        명령문
done
```

```
#!/bin/bash
loop=1
while [ $loop -lt 5 ]
do
        echo This is a while test $loop
        let loop=loop+1
done
```

제어흐름 (8/8)

● 반복 구조 : until

❍ until 문

- 해당조건을 만족할 때까지 **do** 이후에 나오는 명령문을 반복 수행한다.

```
until 조건문
do
        명령문
done
```

```
#!/bin/bash
loop=10
until [ $loop -lt 5 ]
do
        echo This is a until test $loop
        let loop=loop-1
done
```

쉘 프로그래밍 기초

함수, 디버깅

함수

- **함수(Function)**

```
function 함수명 { 명령문 }
```

```
#!/bin/bash
function exam1 {
        exit
}
function exam2 {
        echo Excute Function!!!
}
exam2
exam1
echo Final
```

디버깅

- 디버깅

```
#!/bin/bash -x
```

참고문헌

[1] 이종원, “IT CookBook, 우분투 리눅스(개정판) : 시스템 & 네트워크”, 한빛아카데미, 2018.

[2] 백창우, “유닉스 리눅스 프로그래밍 필수 유틸리티”, 한빛미디어. 2010.

[3] "GNU Operating System", Free Software Foundation(FSF), 2020 of viewing the site, https://www.gnu.org/.