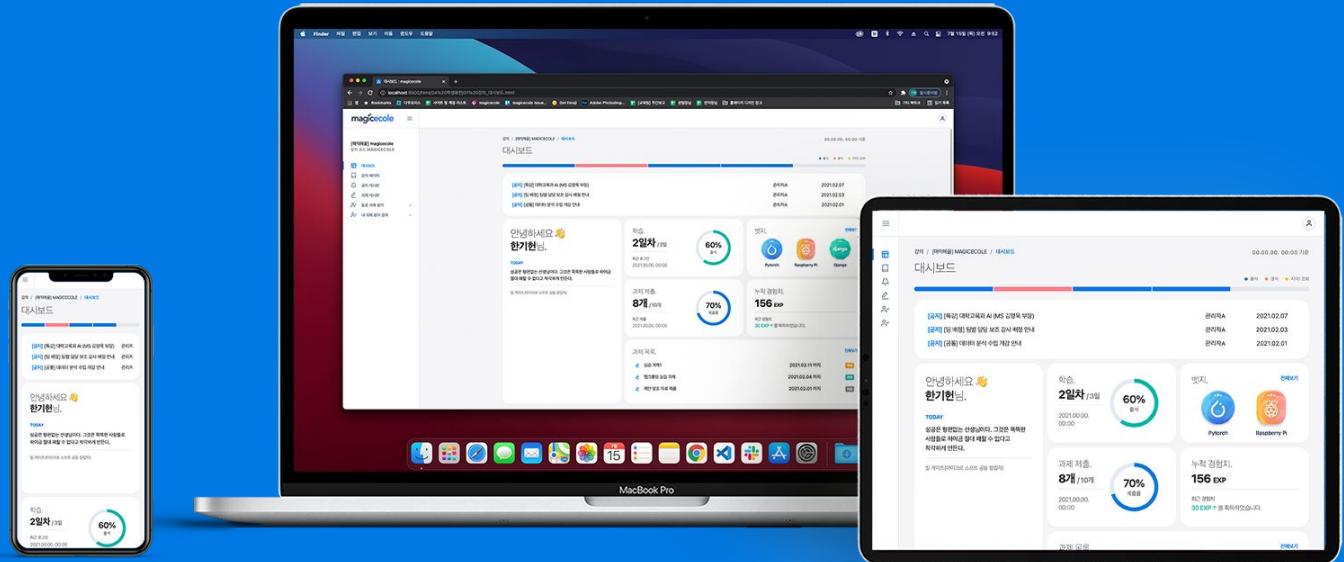


DevOps – Linux Shell



다를 내용

- Shell 기본 개념
- Shell 명령어 실행
- Shell과 변수
- Shell 스크립트 프로그래밍
- 환경설정

Shell 기본 개념

Shell



W Wikipedia
Shell plc - Wikipedia



www.shell.co.kr
The Shell brand | Shell Republic of Korea



SA Scientific American
How are seashells created? Or any other shell,...



Facebook
Shell - Home | Facebook



Pikmin | Fandom
Scrumptious Shell | Pikmin | Fandom



Rustic White & Wood · 재고 있음
Fan Sea Shell – Rustic White & Wood



Google Play
Shell Fleet App - Google Play ...

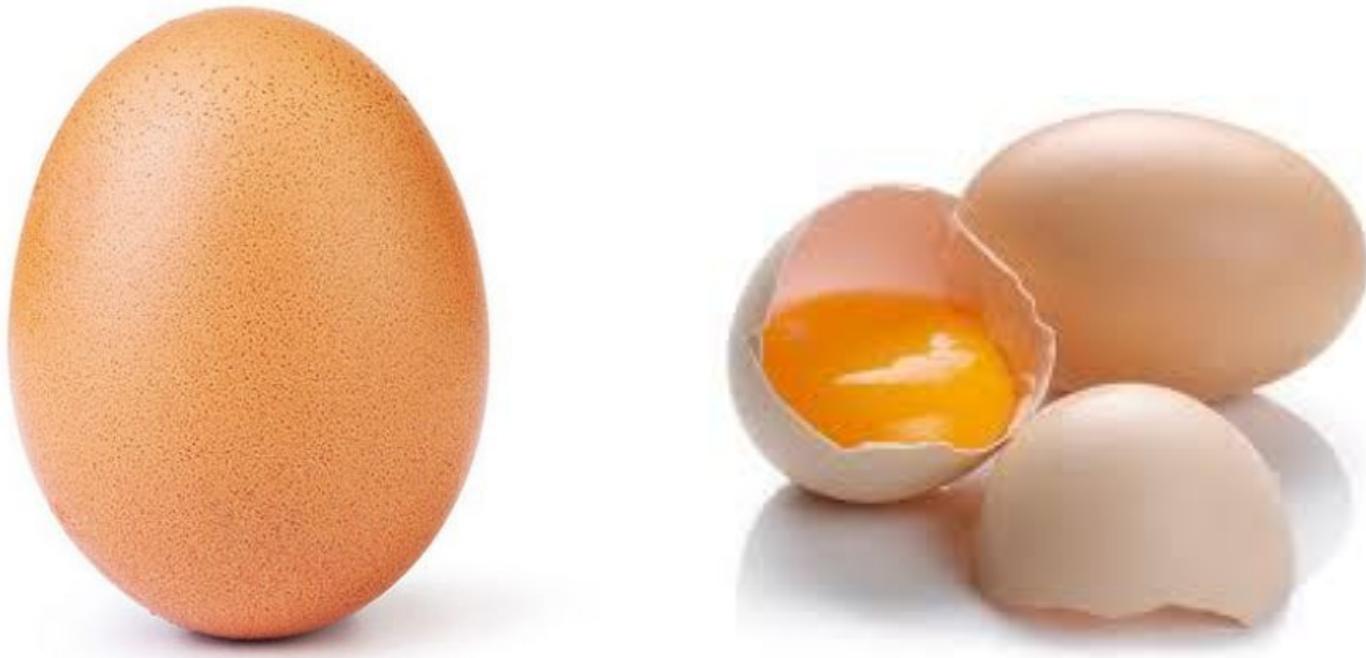


Seeking Alpha
Shell: Superior Energy Portfolio Growth Through...

shell O+ ★★ +

1. 명사 (달걀견과류 등의 딱딱한) **껍데기[껍질]** (→eggshell, nutshell, seashell, tortoiseshell)
2. 명사 고둥 **껍데기[조가비]** 모양의 것
3. 동사 **포격[폭격]**하다
4. 동사 (호두콩 등의) 껍질을 까다

Shell



Shell

우리가 배워야 할 Shell

```
* Canonical Livepatch is available for installation.  
- Reduce system reboots and improve kernel security. Activate at:  
https://ubuntu.com/livepatch
```

230 packages can be updated.

191 updates are security updates.

New release '20.04.5 LTS' available.

Run 'do-release-upgrade' to upgrade to it.

```
Last login: Tue Feb 21 10:33:31 2023 from 221.146.63.213  
root@linux-test:~# ls -al  
total 60  
drwx----- 9 root root 4096 Feb 21 11:55 .  
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..  
-rw------- 1 root root 0 Feb 21 11:55 .bash_history  
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc  
drwx----- 2 root root 4096 Feb 9 11:14 .cache  
drwx----- 3 root root 4096 Oct 6 2020 .gnupg  
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local  
drwxr-xr-x 2 root root 4096 Feb 21 10:50 .nsight  
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile  
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor  
drwx----- 2 root root 4096 Oct 6 2020 .ssh  
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim  
-rw------- 1 root root 4687 Feb 20 14:36 .viminfo  
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server  
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts  
root@linux-test:~#
```

무슨 관계가 있지?

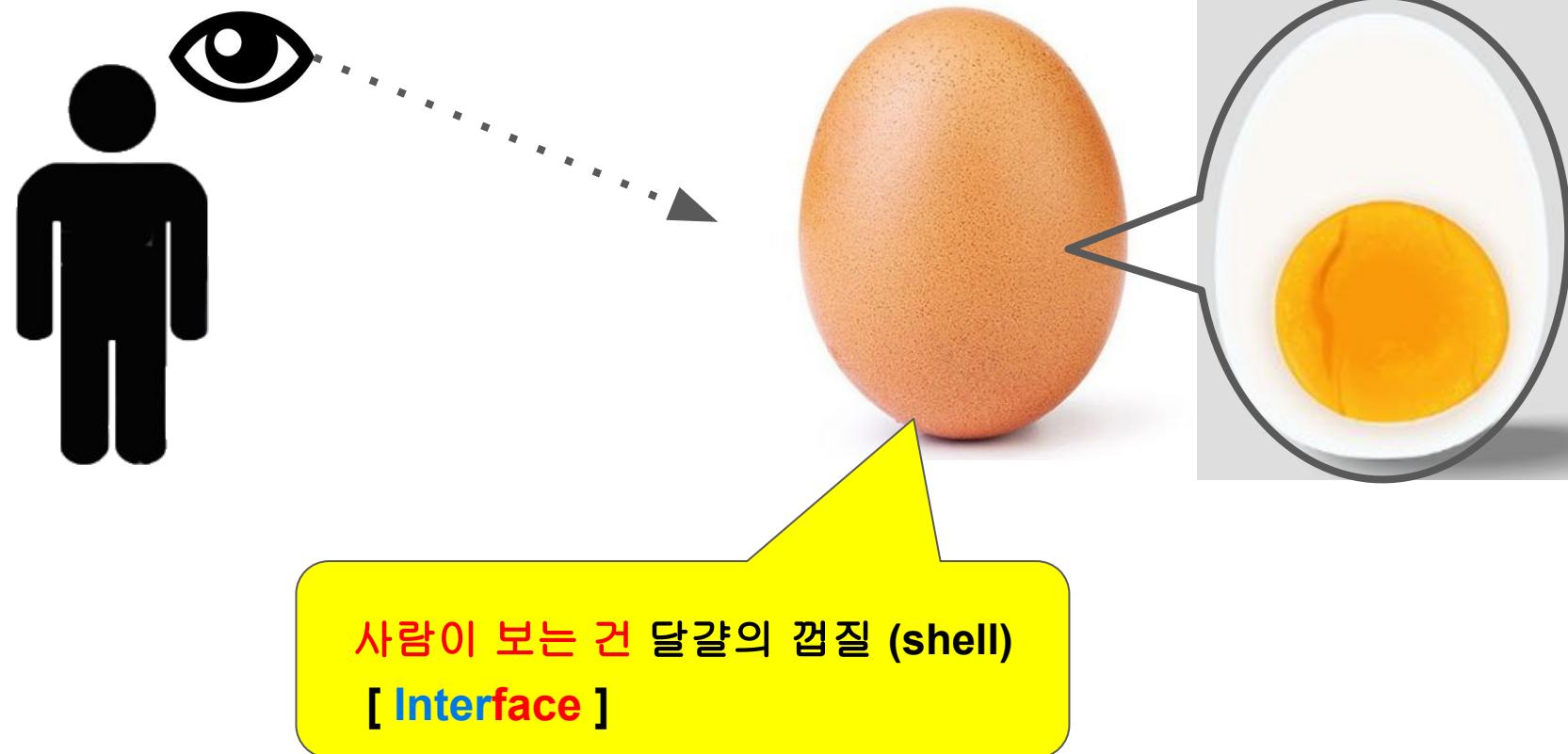


Shell의 정의

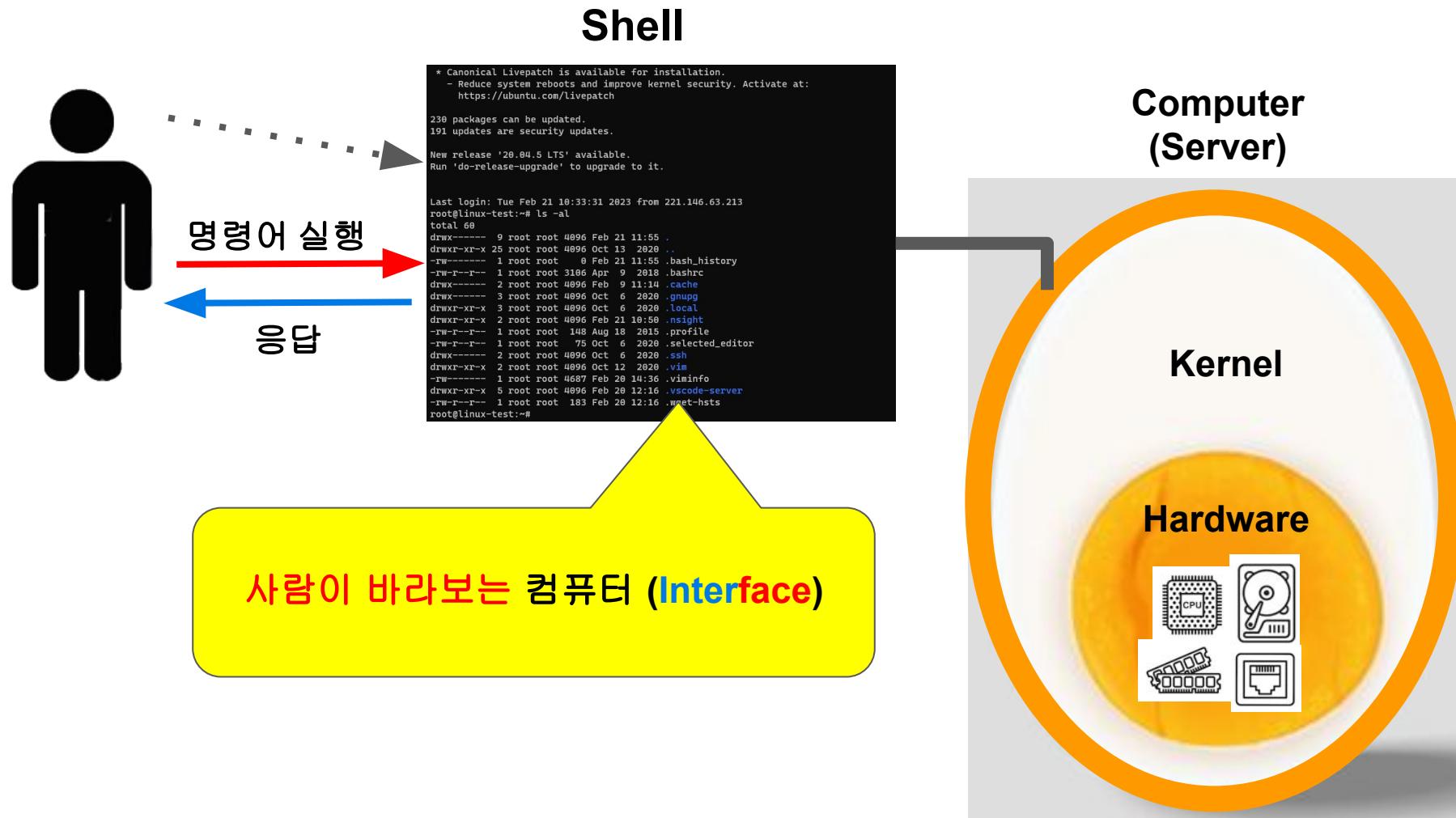
위키피디아 참조

셸(shell, 문화어: 헬, 자령해석프로그램)은 운영 체제 상에서 다양한 운영 체제 기능과 서비스를 구현하는 **인터페이스를 제공하는** 프로그램이다. 셸(껍데기의 영어 단어)은 사용자와 운영 체제의 내부(커널) 사이의 **인터페이스를 감싸는 층**이기 때문에 그러한 이름이 붙었다.

Shell의 특성 - 인터페이스



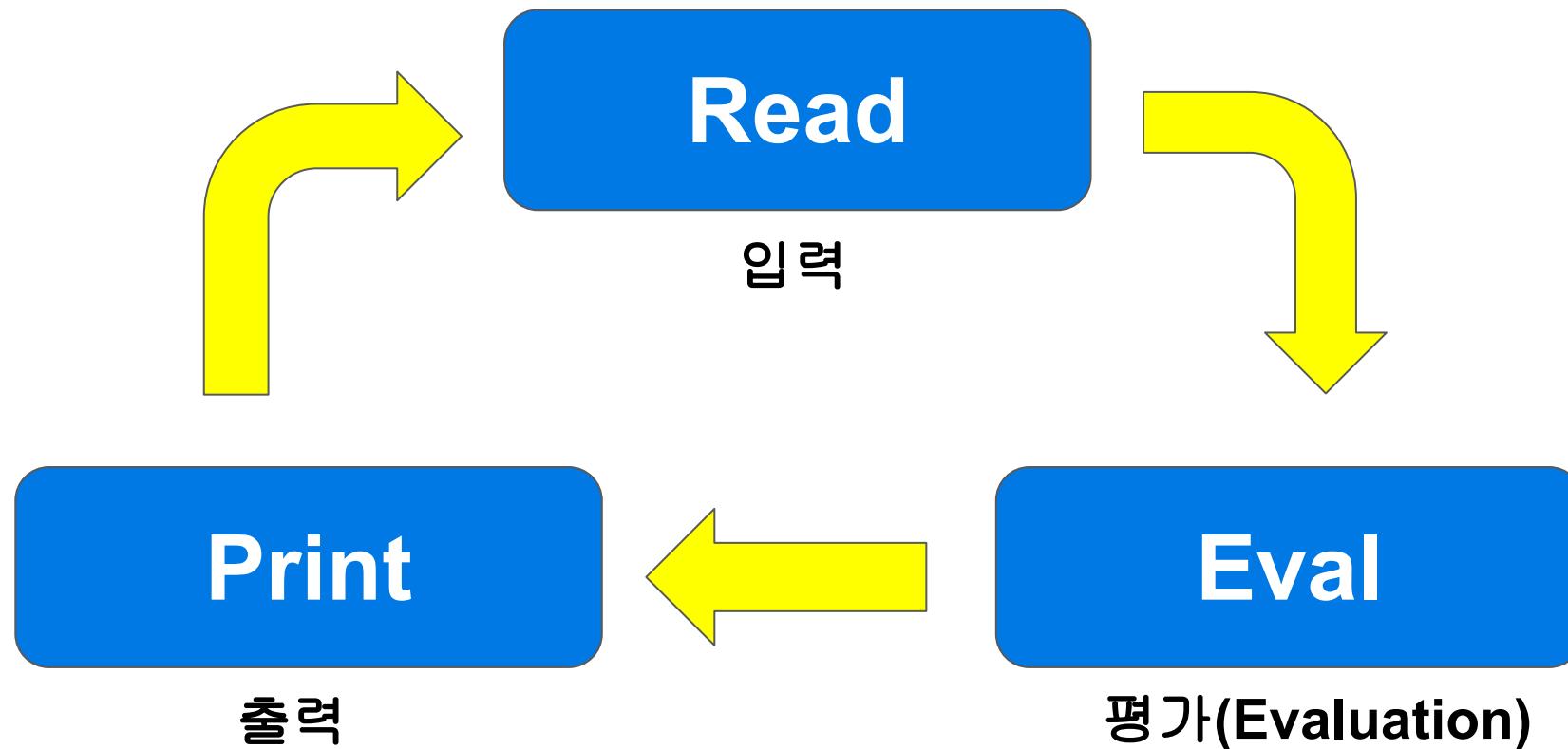
Shell의 특성 - 인터페이스



Shell의 특성 - 인터페이스

REPL (Read - Eval - Print Loop)

‘레플’이라고 읽음



Shell의 특성 - 인터페이스

터미널에서 사용자가 명령어 입력
(Read)

```
root@linux-test:~# ls -al ./
```

Shell 내부 동작

명령어 라인 분석 (Parsing)

명령
ls

옵션
-a, -l

인자
./

...

Shell 내부에 함수로
구현되어 있는가?

ls는 외부 명령어
(별도 실행 파일 존재)

내부(빌트인)
명령어인가?

Yes

Shell 내부에서
함수 호출

Eval(평가) : 명령어 분석 ~ 판단

옵션, 인자 전달

No (외부 명령어)

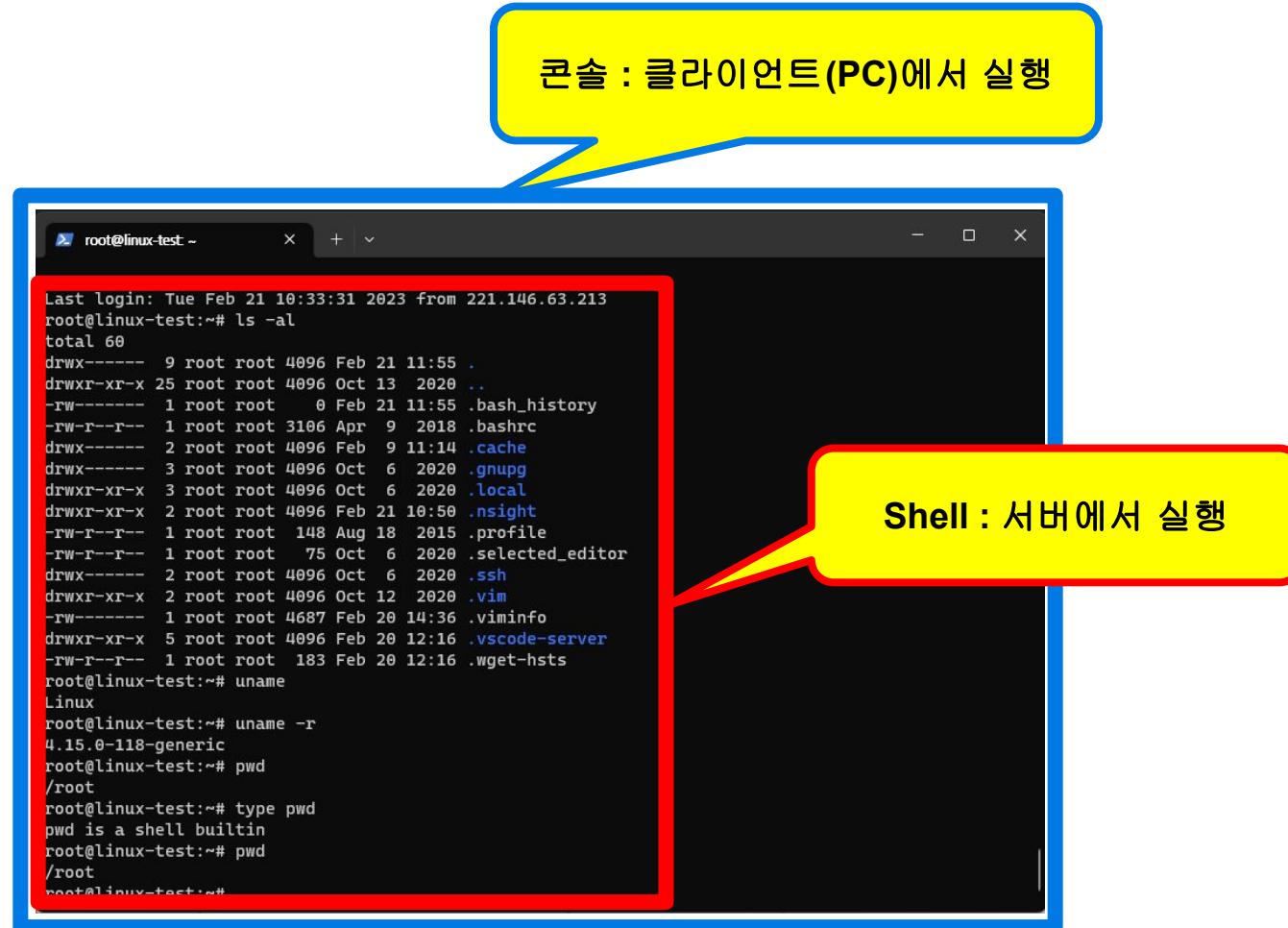
명령 실행 파일 탐색

파일 실행
(별도 프로세스)

종료 (프롬프트 복귀)

Print (출력)

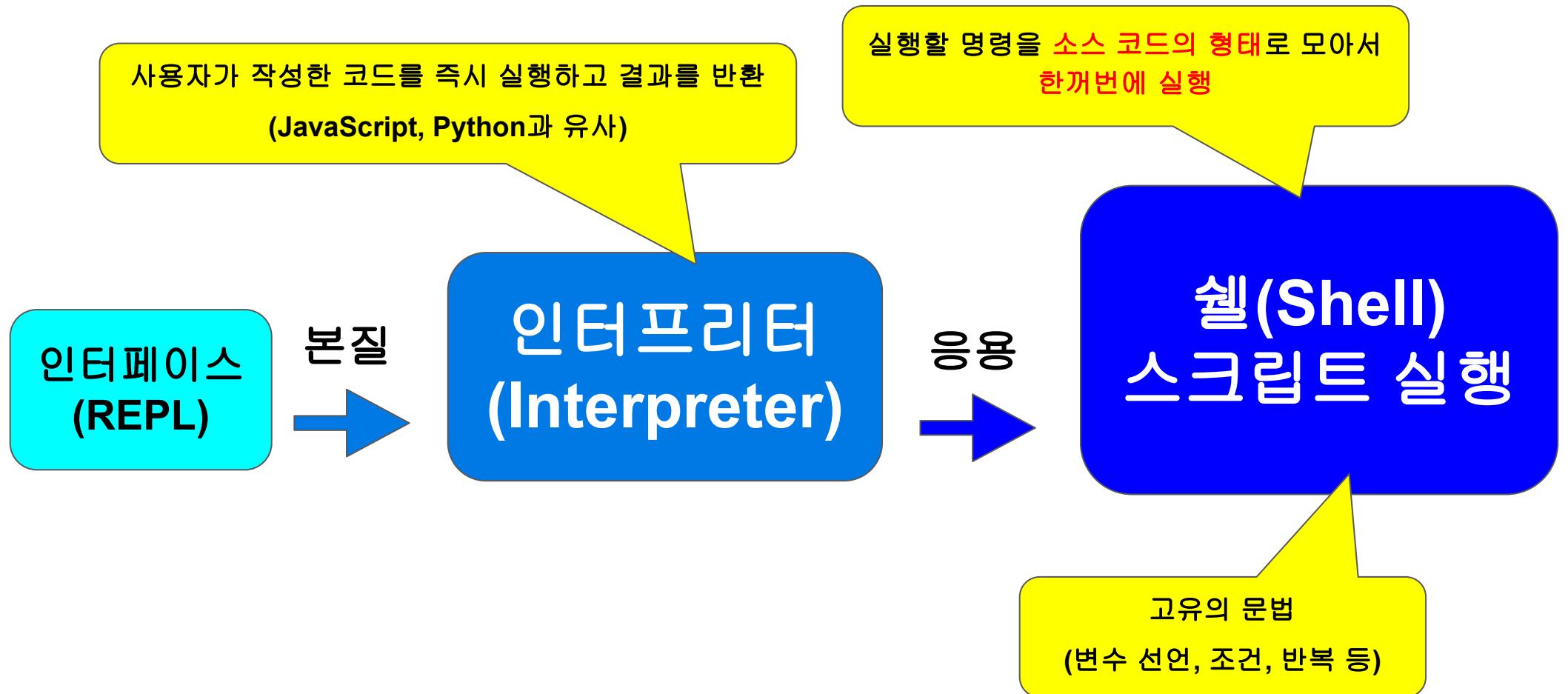
콘솔(터미널) vs Shell



사용자가 콘솔에서 보는 것

=> Shell이 출력하고, 콘솔이 전달받아 모니터로
보여주는 문자열 데이터(스트림)

Shell의 특성 - 인터프리터



REPL과 인터프리터 – JavaScript



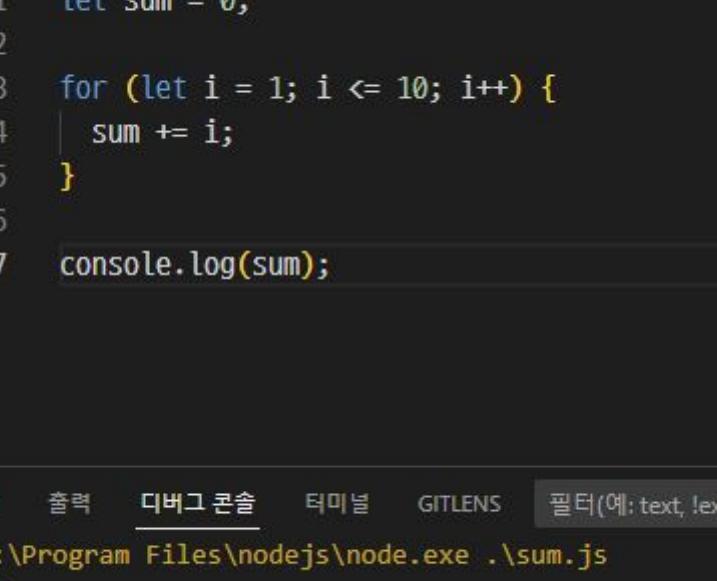
크롬 브라우저 콘솔 (또는 node.js)

The screenshot shows the Google Chrome DevTools interface with the 'Console' tab selected. The left sidebar lists messages: '메시지 1개' (1 message), '사용자 메시지 1개' (1 user message), '오류 없음' (No errors), '경고 없음' (No warnings), '정보 1건' (1 info item), and '상세 메시지 없음' (No detailed messages). The main area displays a REPL session:

```
> let a = 1
< undefined
> a
< 1
> function func(num){
  let com = num;
  console.log(com);
}
< undefined
> func(1)
1
< undefined
>
```

A yellow callout bubble points to the text "사용자가 입력한 라인마다 실행 (REPL)" (Execute on every user input line).

JavaScript 소스 코드(스크립트) 작성



```
js sum.js  U X
js sum.js > ...
1 let sum = 0;
2
3 for (let i = 1; i <= 10; i++) {
4     sum += i;
5 }
6
7 console.log(sum);
```

문제 출력 디버그 콘솔 터미널 GITLENS 필터(예: text, !exclude)

```
C:\Program Files\nodejs\node.exe .\sum.js
55
```

node.js를 통해 코드 실행

REPL과 인터프리터 - Shell

쉘 프롬프트에서 입력한 라인마다 실행
(REPL)

쉘 스크립트를 작성하고 실행

쉘 변수 설정 / 출력

쉘 함수 정의

쉘 함수 호출

```
root@linux-test:~# NUM=1
root@linux-test:~# echo $NUM
1
```

```
root@linux-test:~# print_myname(){
> echo "Hi, my name is $1"
> }
```

```
root@linux-test:~# print_myname Chris
Hi, my name is Chris
root@linux-test:~#
```

`shell_example.sh`

```
#!/bin/sh

NUM=1
echo $NUM

print_myname(){
    echo "Hi, my name is $1"
}

print_myname Chris
```

```
root@linux-test:~# sh shell_example.sh
1
Hi, my name is Chris
```

Shell과 JavaScript 엔진의 차이점

=> JavaScript 엔진 : JS 소스 코드를 실행하기 수단

Shell : 사용자 명령 또는 스크립트를 통한 시스템 관리

REPL과 인터프리터 - Shell

두가지 특성의 쓰임새

인터페이스
(REPL)

사용자 입력에 대한 응답
(사람이 실행)

VS

인터페이스와 인터프리터
두가지 특성을 동시에 지님

인터프리터
(Interpreter)

스크립트의 실행
(사람 또는 시스템이 실행)

예) 환경 설정

리눅스 Shell의 종류

Bourne

Bash
(Bourne Again SHell)
/bin/bash

(우분투) 리눅스 기본 Shell

C

Korn

Dash

Shell 명령어의 실행 1 - 기본

Shell의 명령 종류

내부 명령
(빌트인 명령)

외부 명령

Shell 내부(빌트인) 명령

Shell 내부에 함수 형태로 구현된 명령

```
GNU bash, version 4.4.20(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally. Type 'help' to see this list.
Type 'help name' to find out more about the function 'name'.
Use 'info bash' to find out more about the shell in general.
Use 'man -k' or 'info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.
```

help 명령으로 확인

```
job_spec [&]
(( expression ))
. filename [arguments]
:
[ arg... ]
[[ expression ]]
alias [-p] [name[=value] ... ]
bg [job_spec ...]
bind [-lpsvPSVX] [-m keymap] [-f filename] [-q name] [-u >
break [n]
builtin [shell-builtin [arg ...]]
caller [expr]
case WORD in [PATTERN [| PATTERN]...] COMMANDS ;;;...) esa>
cd [-L|-P [-e]] [-@]] [dir]
command [-pVv] command [arg ...]
compgen [-abcdefgjksuv] [-o option] [-A action] [-G globp>
complete [-abcdefgjksuv] [-pr] [-DE] [-o option] [-A acti>
compopt [-o]+o option] [-DE] [name ...]
continue [n]
coproc [NAME] command [redirections]
declare [-aAfGgilnrdux] [-p] [name[=value] ...]
dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ... | pid ...]
echo [-neE] [arg ...]
enable [-a] [-dnps] [-f filename] [name ...]
eval [arg ...]
exec [-cl] [-a name] [command [arguments ...]] [redirec>
exit [n]
export [-fn] [name[=value] ...] or export -p
false
fc [-e ename] [-lnr] [first] [last] or fc -s [pat=rep] [c>
fg [job_spec]
for NAME [in WORDS ... ] ; do COMMANDS; done
for (( exp1; exp2; exp3 )); do COMMANDS; done
function name { COMMANDS ; } or name () { COMMANDS ; }
getopts optstring name [arg]
hash [-lr] [-p pathname] [-dt] [name ...]
help [-dms] [pattern ...]
```

```
history [-c] [-d offset] [n] or history -anrw [filename]>
if COMMANDS; then COMMANDS; [ elif COMMANDS; then COMMANDS;
jobs [-lnprs] [jobspec ...] or jobs -x command [args]
kill [-s sigspec | -n signum | -sigspec] pid | jobspec .>
let arg [arg ...]
local [option] name[=value] ...
logout [n]
mapfile [-d delim] [-n count] [-O origin] [-s count] [-t>
popd [-n] [+N | -N]
printf [-v var] format [arguments]
pushd [-n] [+N | -N | dir]
pwd [-LP]
read [-ers] [-a array] [-d delim] [-i text] [-n nchars] >
readarray [-n count] [-O origin] [-s count] [-t] [-u fd]>
readonly [-aAf] [name[=value] ...] or readonly -p
return [n]
select NAME [in WORDS ... ;] do COMMANDS; done
set [-abefhkmnpstuvwxyzBCHP] [-o option-name] [--] [arg ...]
shift [n]
shopt [-pqsu] [-o] [optname ...]
source filename [arguments]
suspend [-f]
test [expr]
time [-p] pipeline
times
trap [-lp] [[arg] signal_spec ...]
true
type [-afptP] name [name ...]
typeset [-aAfGgilnrdux] [-p] name[=value] ...
ulimit [-SHabcdefiklmnpqrstuvwxyzPT] [limit]
umask [-p] [-S] [mode]
unalias [-a] name [name ...]
unset [-f] [-v] [-n] [name ...]
until COMMANDS; do COMMANDS; done
variables - Names and meanings of some shell variables
wait [-n] [id ...]
while COMMANDS; do COMMANDS; done
{ COMMANDS ; }
```

외부 명령

별도의 실행 가능한 바이너리 파일로 구현(프로그램)

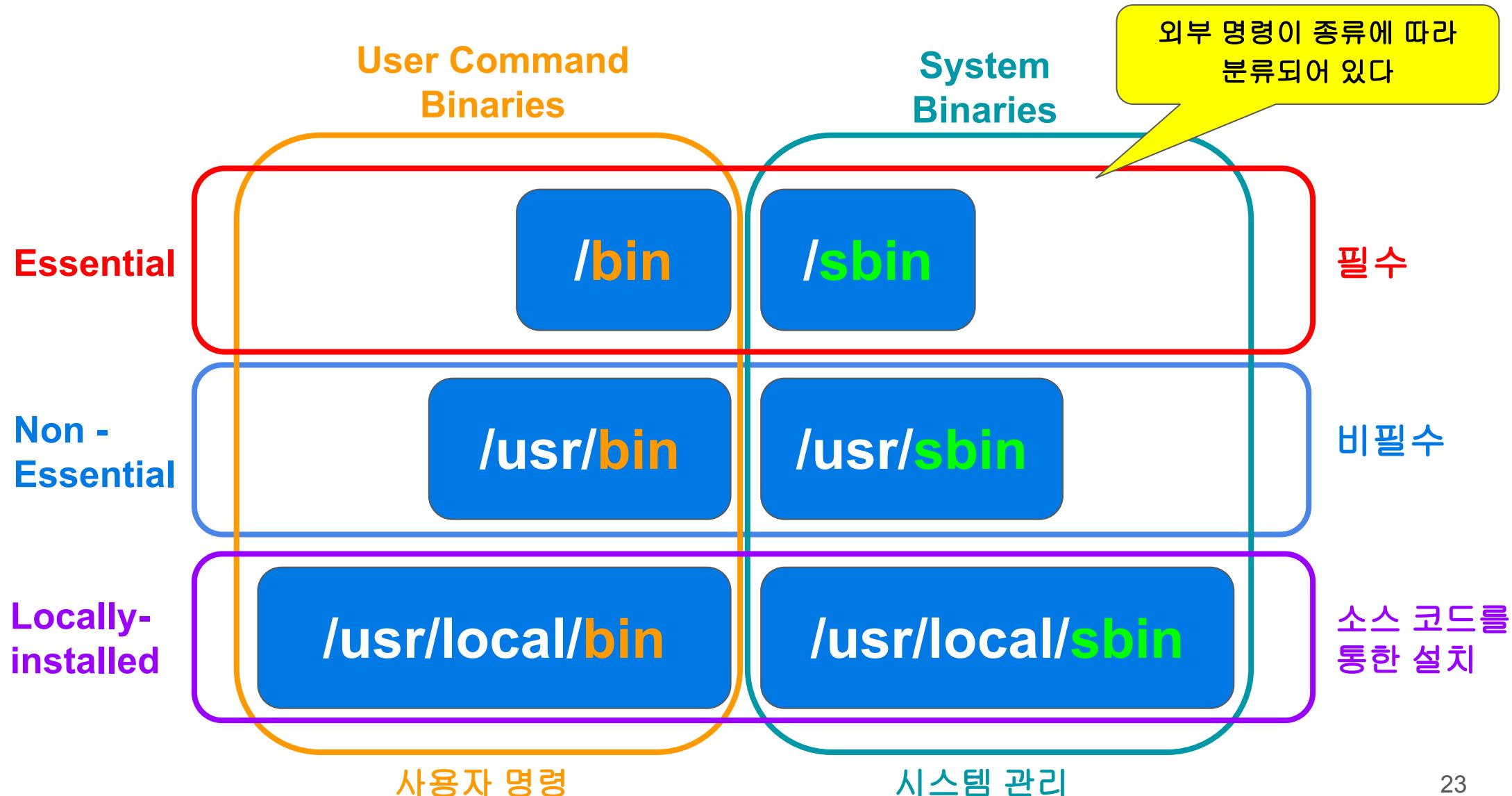
파일이므로 ls 명령으로
확인 가능

```
root@linux-test:~# ls /bin
```

bash	cp	kbd_mode	netcat	readlink		tar
btrfs	cpio	kill	netstat	red		tempfile
btrfsck	dash	kmod	networkctl	rm		touch
btrfs-debug-tree	date	less	nisdomainname	rmdir		true
btrfs-find-root	dd	lessecho	ntfs-3g	rnano		udevadm
btrfs-image	df	lessfile	ntfs-3g.probe	run-parts		unlockmgr_server
btrfs-map-logical	dir	lesskey	ntfscat	sed		umount
btrfs-select-super	dmesg	lesspipe	ntfscluster	setfacl		uname
btrfstune	dnsdomainname	ln	ntfscmp	setfont		uncompress
btrfs-zero-log	domainname	loadkeys	ntfsallocate	setupcon		unicode_start
bunzip2	dumpkeys	login	ntfsfix	sh		vdir
busybox	echo	logind	ntfsinfo	sh.distrib		wdctl
bzcat	ed	lowntfs-3g	ntfsls	sleep		which
bzcmp	egrep	ls	ntfsmove	ss		whiptail
bzdiff	false	lsblk	ntfsrecover	static-sh		ypdomainname
bzegrep	fgconsole	lsmod	ntfssecaudit	stty		zcat
bzexe	fgrep	mkdir	ntfstruncate	su		zcmp
bzfgrep	findmnt	mkfs.btrfs	ntfsusermap	sync		zdiff
bzgrep	fsck.btrfs	mknod	ntfswipe	systemctl		zegrep
bzip2	fuser	mktemp	open	systemd		zfgrep
bzip2recover	fusermount	more	openvt	systemd-ask-password		zforce
bzless	getfacl	mount	pidof	systemd-escape		zgrep
bzmore	grep	mountpoint	ping	systemd-hwdb		zless
cat	gunzip	mt	ping4	systemd-inhibit		zmore
chacl	gzexe	mt-gnu	ping6	systemd-machine-id-setup		znew
chgrp	gzip	mv	plymouth	systemd-notify		
chmod	hostname	nano	ps	systemd-sysusers		
chown	ip	nc	pwd	systemd-tmpfiles		
chvt	journalctl	nc.openbsd	rbash	systemd-tty-ask-password-agent		

외부 명령 - 실행 파일 경로

파일 계층 표준(File Hierarchy Standard)에 정의



외부 명령

내부 명령과 외부 명령의 차이

내부 명령

```
root@linux-test:~# type cd  
cd is a shell builtin
```

type

기능: 명령의 타입 정보 출력
형식: type [옵션] [명령]

외부 명령

```
root@linux-test:~# type cp  
cp is /bin/cp
```

파일의 종류 확인

```
root@linux-test:~# file /bin/cp  
/bin/cp: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,  
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=9720743c3  
b9fbf61b6f38b7568c10a6dbed05a61, stripped
```

서버의 CPU 타입 확인

```
root@linux-test:~# lscpu  
Architecture: x86_64  
CPU op-mode(s): 32-bit, 64-bit  
Byte Order: Little Endian
```

cp 명령은 이 서버에서 실행
가능하다!

Shell의 기본 출력 명령

● echo

- **기능** : 화면에 한 줄의 문자열을 출력
- **형식** : echo [옵션] [문자열]

1. 화면에 “Hello World” 문자열 출력 : echo Hello World
echo “Hello World”

```
root@linux-test:~# echo Hello World
Hello World
root@linux-test:~# echo "Hello World"
Hello World
```

● printf

- **기능** : C언어의 printf 함수의 형식으로 문자열 출력
- **형식** : printf [형식] [인자]

1. 화면에 C언어 printf 함수 형식으로 “My name is Linux” 문자열 출력 : print “My name is %s\n” Linux

```
root@linux-test:~# printf "My name is %s\n" Linux
My name is Linux.....
```

명령어 히스토리

- **history**

- **기능** : 명령 입력 기록을 출력
- **형식** : **history [숫자]**

숫자를 지정하지 않을 경우
전체 목록 출력

1. 최근에 입력한 마지막 10개 명령어 목록을 출력 : **history 10**

```
root@linux-test:~# history 10
31 echo \"Hello World\"
32 printf "Hello World \n"
33 printf "My name is %s\n" Linux
34 printf --help
35 printf
36 printf "aha"
37 printf "My name is %s\n" $USER
38 printf "My name is %s\n" "Linux"
39 history
40 history 10
```

목록 번호

명령어

히스토리 목록 번호를 통한 이전 명령 실행

```
root@linux-test:~# !33
printf "My name is %s\n" Linux
My name is Linux
```

프롬프트에서 키보드의   키로도

이전 명령 조회 가능

명령어 앤리어스

- **alias**

- **기능** : 기존 명령을 대신하는 별칭 지정
- **형식** : **history [이름='명령']**

1. 현재 지정된 앤리어스 출력 : **alias**

```
root@linux-test:~# alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
root@linux-test:~#
```

ls -alF 명령에 대한
별칭을 II로 지정

앤리어스를 통한 명령 실행

```
root@linux-test:~# ll
total 100
drwx----- 9 root root 4096 Mar  5 10:28 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rwxr-xr-x  1 root root 8640 Mar  2 13:54 a.out*
-rw-------  1 root root     0 Mar  5 10:20 .bash_history
-rw-r--r--  1 root root   33 Feb 22 14:53 .bash_logout
-rw-r--r--  1 root root 3106 Apr  9 2018 .bashrc
drwx-----  2 root root 4096 Feb  9 11:14 .cache/
drwx-----  3 root root 4096 Oct  6 2020 .gnupg/
drwxr-xr-x  3 root root 4096 Oct  6 2020 .local/
drwxr-xr-x  2 root root 4096 Mar  5 09:56 .nsight/
-rw-r--r--  1 root root  999 Mar  2 13:55 output.txt
-rw-r--r--  1 root root  148 Aug 18 2015 .profile
-rw-r--r--  1 root root   75 Oct  6 2020 .selected_editor
-rw-r--r--  1 root root  259 Mar  3 18:20 shell_ex2.sh
-rw-r--r--  1 root root  100 Mar  3 14:02 shell_example.sh
drwx-----  2 root root 4096 Oct  6 2020 .ssh/
-rw-r--r--  1 root root  905 Mar  2 13:54 test.c
drwxr-xr-x  2 root root 4096 Oct 12 2020 .vim/
-rw-------  1 root root 13823 Mar  5 10:28 .viminfo
drwxr-xr-x  5 root root 4096 Feb 20 12:16 .vscode-server/
-rw-r--r--  1 root root  183 Feb 20 12:16 .wget-hsts
```

명령어 앤리어스

- **alias**

2. date 명령에 대한 앤리어스 지정 : **alias date_fmt='date "+%Y-%m-%d %H:%M:%S"'**

```
root@linux-test:~# alias date_fmt='date "+%Y-%m-%d %H:%M:%S"'  
root@linux-test:~# date_fmt  
2023-03-05 11:08:56  
root@linux-test:~# date  
Sun Mar  5 11:09:04 KST 2023  
root@linux-test:~#
```

date 명령의 출력 포맷을 다르게
하도록 앤리어스 지정

앨리어스 실행

기본 특수 문자

단순 문자가 아닌,
Shell에서 특수한 기능을 갖는 문자

* (별표) : 0개 이상의 문자로 대체

- 파일명이 .bash로 시작하는 모든 파일의 목록을 출력 : ls .bash*

```
root@linux-test:~# ls .bash*
```

“.bash”로 시작하는
문자열로 대체

.bash_history .bash_logout .bashrc

- 파일명이 .conf로 끝나는 모든 파일을 /etc 디렉토리에서 찾음 : find /etc -name *.conf

```
root@linux-test:~# find /etc -name *.conf
```

“.conf”로 끝나는
문자열로 대체

```
/etc/deluser.conf  
/etc/vmware-tools/vgauth.conf  
/etc/vmware-tools/tools.conf  
/etc/ucf.conf  
/etc/sysctl.d/10-zeropage.conf
```

기본 특수 문자

“ ” , ‘ ’ : 문자열 만들기, 특수 문자 기능 제거

큰따옴표
(Double Quote)

작은따옴표
(Single Quote)

‘ ’ : 내부의 모든 특수 문자 기능 제거

“ ” : \$, ` (역따옴표), \ 문자 제외

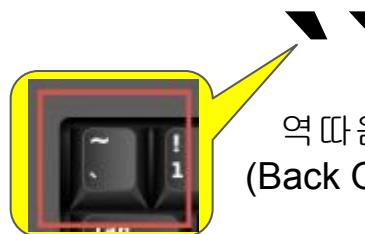
1. ls .bash*, ls '.bash*', ls “.bash*” 의 차이

```
root@linux-test:~# ls .bash*           특수 문자 적용
.bash_history .bash_logout .bashrc
root@linux-test:~# ls '.bash*'         문자열 .bash*
ls: cannot access '.bash*': No such file or directory
root@linux-test:~# ls ".bash*"        문자열 .bash*
ls: cannot access '.bash*': No such file or directory
```

2. echo \$\$SHELL, echo '\$\$SHELL', echo “\$\$SHELL” 의 차이

```
root@linux-test:~# echo $$SHELL          특수 문자 적용
/bin/bash
root@linux-test:~# echo '$$SHELL'        => SHELL이라는 이름의 환경 변수 값 출력
$SHELL
root@linux-test:~# echo "$$SHELL"        문자열 $SHELL
/bin/bash                               특수 문자 적용
```

기본 특수 문자



': 문자열을 명령으로 해석, 명령 실행 결과로 치환

역따옴표
(Back Quote)

\$**(명령)**도 같은 역할

1. date 명령의 실행 결과를 문자열에 포함하여 출력 : echo "Today is `date`"

```
root@linux-test:~# echo "Today is `date`"  
Today is Thu Feb 23 17:03:58 KST 2023
```

date 명령의 실행 결과로 치환

2. 1과 동일 : echo "Today is \$(date)"

```
root@linux-test:~# echo "Today is $(date)"  
Today is Thu Feb 23 17:26:51 KST 2023
```

`date`와 동일

기본 특수 문자

\ (또는 ) : 바로 뒤에 오는 문자의 특수 기능 제거(Escape 문자)
긴 명령을 여러 행에 입력 가능(Line Continuation)

1. * 문자의 특수 기능 삭제(일반 문자화) : ls .bash*

```
root@linux-test:~# ls .bash\*
ls: cannot access '.bash*': No such file or directory
```

2. space 문자의 특수 기능 삭제 : touch Hello\ World

```
root@linux-test:~# touch Hello\ World
root@linux-test:~# ls
'Hello World'
```

space앞에 \를 입력하여
특수 기능 제거 (단순 빈칸)

빈 칸이 들어간 파일명 생성

Shell에서 space
- 명령, 옵션, 인자를 나누는 기능

기본 특수 문자

\ (또는 \\)

3. find /etc -name *.conf 명령을 여러 행에 입력

```
root@linux-test:~# find \
> /etc \
> -name \
> *.conf
```

\ 문자로 Line Continuation 적용
=> \ 뒤에 입력하는 Enter의 기능(명령 실행) 제거

```
/etc/deluser.conf
/etc/vmware-tools/vgauth.conf
/etc/vmware-tools/tools.conf
/etc/ucf.conf
/etc/sysctl.d/10-zeropage.conf
/etc/sysctl.d/10-network-security.conf
```

명령어를 한줄로 입력할 때와 동일

Shell 명령어의 실행 2 - 응용

그룹 명령 실행

하나의 명령 행에 여러 명령어를 동시에 실행

명령1 ; 명령2 : 한 줄에 여러 명령을 나열, **입력한 순서로 실행**

명령1 && 명령2 : 명령1이 성공하면 명령2 실행
(논리적 AND)

논리값이 True

어떤 상황에서 명령2를 실행할 것인가?

명령1 || 명령2 : 명령1이 실패하면 명령2 실행
(논리적 OR)

논리값이 False

조건부 연산자 특성 응용

A	B	A AND B	A OR B	NOT A
A	B	A && B	A B	!A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

그룹 명령 실행

조건부 연산자 설명 (by ChatGPT)



||와 &&는 조건부 연산자(Conditional Operator)입니다.

오른쪽 피연산자의 평가
결과에 상관 없음

||는 "OR" 연산자로, 왼쪽 피연산자가 참(True)이면 오른쪽 피연산자를 평가하지 않고 전체 식이 참이 되며, 왼쪽 피연산자가 거짓(False)이면 오른쪽 피연산자를 평가하여 그 결과에 따라 전체 식이 참(True) 또는 거짓(False)이 됩니다.

&&는 "AND" 연산자로, 왼쪽 피연산자가 거짓(False)이면 오른쪽 피연산자를 평가하지 않고 전체 식이 거짓이 되며, 왼쪽 피연산자가 참(True)이면 오른쪽 피연산자를 평가하여 그 결과에 따라 전체 식이 참(True) 또는 거짓(False)이 됩니다.

따라서 이러한 연산자를 사용하여 뒤에 붙는 명령을 실행할지 말지 결정하는 것은, 해당 연산자의 왼쪽 피연산자와 오른쪽 피연산자가 참/거짓 여부에 따라 달라집니다.

그룹 명령 실행 - 실습

1. ls /, date, ls -al 명령을 순서대로 실행 : ls / ; date ; ls -al

```
root@linux-test:~# ls / ; date ; ls -al
bin   home1          lib64      proc    srv  vmlinuz
boot  initrd.img     lost+found  root    sys  vmlinuz.old
dev   initrd.img.old media      run     tmp
etc   lib            mnt       sbin   usr
home  lib32         opt       snap   var

Fri Feb 24 17:23:46 KST 2023

total 64
drwx----- 9 root root 4096 Feb 24 17:23 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 24 09:24 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
drwxr-xr-x 2 root root 4096 Feb 24 12:17 .nsight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 7455 Feb 22 14:53 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 wget-hsts
```

date 명령 실행

ls / 명령 실행

ls -al 명령 실행

그룹 명령 실행 - 실습

2. 명령1 && 명령2 실행 비교

2-1) date && ls /

```
root@linux-test:~# date && ls /
Fri Feb 24 17:45:37 KST 2023
bin   home1      lib64      proc    srv  vmlinuz
boot  initrd.img  lost+found  root    sys  vmlinuz.old
dev   initrd.img.old media      run     tmp
etc   lib        mnt       sbin    usr
home  lib32     opt       snap    var
root@linux-test:~#
```

date 명령이 정상 실행
=> 뒤에 있는 ls / 명령도 실행

2-2) data && ls /

```
root@linux-test:~# data && ls /
Command 'data' not found, did you mean:

  command 'date' from deb coreutils
  command 'dat' from deb liballegro4-dev

Try: apt install <deb name>
root@linux-test:~#
```

data는 없는 명령

data 명령은 없으므로 에러 발생
=> 뒤에 있는 ls / 명령은 실행되지 않음

그룹 명령 실행 - 실습

3. 명령1 || 명령2 실행 비교

3-1) date || ls /

```
root@linux-test:~# date || ls /
Fri Feb 24 17:48:34 KST 2023
root@linux-test:~#
```

date 명령이 정상 실행

=> 뒤에 있는 ls / 명령은 실행되지 않음

3-2) data || ls /

data는 없는 명령

```
root@linux-test:~# data || ls /
```

Command 'data' not found, did you mean:

command 'date' from deb coreutils
command 'dat' from deb liballegro4-dev

Try: apt install <deb name>

bin	home1	lib64	proc	srv	vmlinuz
boot	initrd.img	lost+found	root	sys	vmlinuz.old
dev	initrd.img.old	media	run	tmp	
etc	lib	mnt	sbin	usr	
home	lib32	opt	snap	var	

data 명령은 없으므로 에러 발생

=> 뒤에 있는 ls / 명령이 실행됨

```
root@linux-test:~#
```

표준 입출력 (Standard In/Output) 장치

- 컴퓨터와 상호작용 할 수 있는 **가장 기본적인 장치** -



사람이 명령을 전달하고
결과를 받을 수 있는



최소한의 장치

표준 입출력 - 장치 (Device)

쉽게 말하면...

```
https://ubuntu.com/livepatch
```

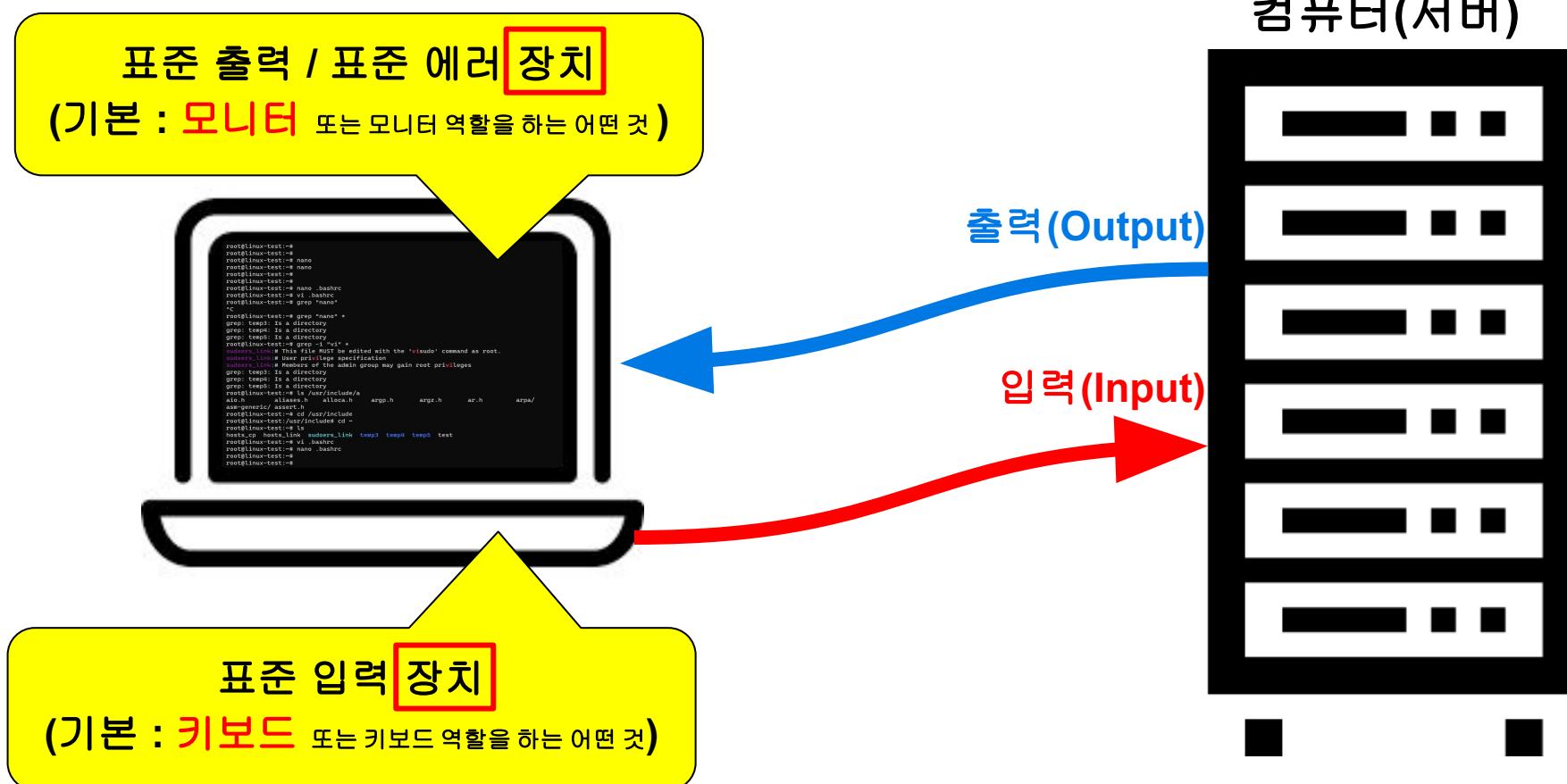
```
230 packages can be updated.  
191 updates are security updates.  
  
New release '20.04.5 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.
```

```
Last login: Mon Feb 27 09:44:34 2023 from 221.146.63.213  
root@linux-test:~# ls  
root@linux-test:~# ls -al  
total 64  
drwx----- 9 root root 4096 Feb 27 10:28 .  
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..  
-rw------- 1 root root 0 Feb 27 10:28 .bash_history  
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout  
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc  
drwx----- 2 root root 4096 Feb 9 11:14 .cache  
drwx----- 3 root root 4096 Oct 6 2020 .gnupg  
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local  
drwxr-xr-x 2 root root 4096 Feb 27 09:17 .nsight  
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile  
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor  
drwx----- 2 root root 4096 Oct 6 2020 .ssh  
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim  
-rw------- 1 root root 7455 Feb 22 14:53 .viminfo  
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server  
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts  
root@linux-test:~#
```

이 환경(CLI)에서 사용할 수 있는
입출력 장치

표준 입출력 - 장치(Device)

장치 : 컴퓨터(본체)가 입/출력 하는 대상

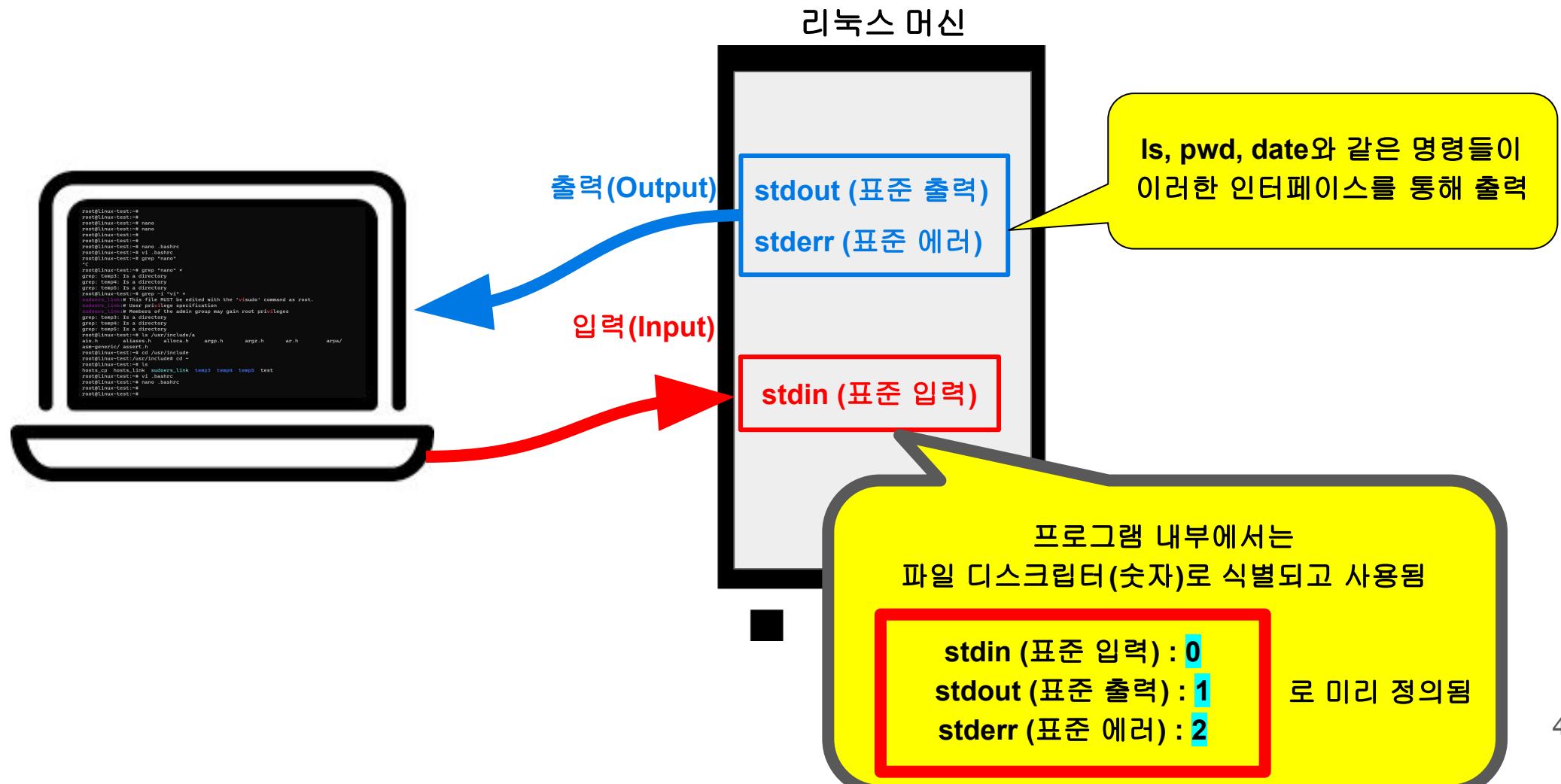


표준 입출력 - 인터페이스

표준 입력 / 출력 / 에러

- 표준 입출력 장치를 위한 (리눅스의) 인터페이스

리눅스의 프로그램이
데이터를 보내고 받기 위한 **창구**



표준 입출력 - 인터페이스

표준 입력 / 출력 / 에러 (stdin, stdout, stderr)를 사용한 코드 예제 (C언어)

```
#include <unistd.h>
#include <stdio.h>
#include <string.h>

#define STDIN_FD 0
#define STDOUT_FD 1
#define STDERR_FD 2

int main() {
    char input[100];
    int num, ret;

    ret = write(STDOUT_FD, "Enter a number: ", 16);
    if (ret < 0) {
        write(STDERR_FD, "Error: failed to write output\n", 31);
        return 1;
    }

    ret = read(STDIN_FD, input, sizeof(input));
    if (ret < 0) {
        write(STDERR_FD, "Error: failed to read input\n", 28);
        return 1;
    }
    input[ret-1] = '\0';

    ret = sscanf(input, "%d", &num);
    if (ret != 1) {
        char buf[100];
        sprintf(buf, "Error: invalid input (%s)\n", input);
        write(STDERR_FD, buf, strlen(buf));
        return 1;
    }

    char output[100];
    sprintf(output, "The square of %d is %d\n", num, num*num);
    write(STDOUT_FD, output, strlen(output));

    return 0;
}
```

stdin, stdout, stderr에 대한
파일 디스크립터 값 정의 (0, 1, 2)

stdout (1) 을 통한
문자열 출력

stdin (0) 을 통한
키보드 입력

stderr (2) 을 통한
에러 출력

stdout

코드 실행 화면

```
root@linux-test:~# ./a.out
Enter a number: 3 stdin
The square of 3 is 9 stdout
root@linux-test:~# ./a.out
Enter a number: stdout
Error: invalid input () stderr
root@linux-test:~#
```

ls, pwd 등의 명령도
결국 같은 방식으로 입출력

표준 입출력 - 표준 입력/출력/에러의 변경 ?

표준 입력으로 받은 문자열의
행, 단어, 바이트 수를 출력
(Ctrl + D로 종료)

```
root@linux-test:~# wc
Hi, How are you?
I'm fine, thank you.
2      8     38
```

행 단어 바이트

표준 입력 이용
(키보드 -> stdin)

```
root@linux-test:~# ls -al
total 68
drwx----- 9 root root 4096 Feb 27 10:28 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw------- 1 root root 68 Feb 27 11:19 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr  9 2018 .bashrc
drwx----- 2 root root 4096 Feb  9 11:14 .cache
drwx----- 3 root root 4096 Oct  6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct  6 2020 .local
drwxr-xr-x 2 root root 4096 Feb 27 12:14 .nsight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct  6 2020 .selected_editor
drwx----- 2 root root 4096 Oct  6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw------- 1 root root 7455 Feb 22 14:53 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 wget-hsts
```

표준 출력 이용
(stdout -> 모니터)

```
root@linux-test:~# ls kkk
ls: cannot access 'kkk': No such file or directory
```

표준 에러 이용
(stderr -> 모니터)

입출력의 대상을
바꿀 수는 없을까?

표준 입출력 - 표준 입력/출력/에러의 변경 ?

예를 들어...

```
root@linux-test:~# ls -al
```

```
total 68
drwx----- 9 root root 4096 Feb 27 10:28 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 68 Feb 27 11:19 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
drwxr-xr-x 2 root root 4096 Feb 27 12:14 .nslight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 7455 Feb 22 14:53 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 wget-hsts
```

표준 출력 / 표준 에러는
기본적으로 휘발성!!
(Shell이 종료되면 사라짐)

출력되는 내용에 대해
별도의 저장이 필요한 경우

```
root@linux-test:~# ls kkk
```

```
ls: cannot access 'kkk': No such file or directory
```

Redirection

- 입출력의 방향을 다시 정한다 -

리다이렉션(Redirection)의 의미

입력
리다이렉션

```
root@linux-test:~# wc
Hi, Ho      입력
I'm fine.   u.
              2      8     38
```

표준 입력을 파일로 변경

File

- 키보드가 아닌 파일의 내용을 입력 -

출력
리다이렉션

```
root@linux-test:~# ls -al
total 68
drwx----- 9 root root 4096 Feb 27 10:28 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root  68 Feb 27 11:19 .bash_history
-rw-r--r-- 1 root root  33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr  9  2018 .bashrc
drwx----- 2 root root 4096 Feb  9 11:14 .cache
drwx----- 3 root    2020 .gnupg
drwxr-xr-x  3 root    2020 .local
drwxr-xr-x  2 root root 4096 Feb 27 12:14 .nsight
-rw-r--r--  1 root root 148 Aug 18  2015 .profile
-rw-r--r--  1 root root  75 Oct  6  2020 .selected_editor
drwx----- 2 root root 4096 Oct  6  2020 .ssh
drwxr-xr-x  2 root root 4096 Oct 12 2020 .vim
-rw-----  1 root root 7455 Feb 22 14:53 .viminfo
drwxr-xr-x  5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r--  1 root root 183 Feb 20 12:16 wget-hsts
```

표준 출력을 파일로 변경

File

- 모니터가 아닌 파일로 출력 -

에러
리다이렉션

```
root@linux-test:~# ls kkk
ls: cannot access 'kkk': No such file or directory
```

표준 에러를 파일로 변경

File

에러

출력 리다이렉션

명령 > 파일 : 명령의 표준 출력을 지정한 파일로 변경 (파일이 없으면 생성, 있으면 덮어쓰기)
(명령 1> 파일) (정상 동작) (Overwrite)

stdout

1. ls -al 명령의 실행 결과를 ls.txt 파일로 출력 : ls -al > ls.txt
ls -al 1> ls.txt

```
root@linux-test:~# ls -al > ls.txt
root@linux-test:~# cat ls.txt
total 68
drwx----- 9 root root 4096 Feb 27 15:09 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw------- 1 root root 68 Feb 27 11:19 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr  9 2018 .bashrc
drwx----- 2 root root 4096 Feb  9 11:14 .cache
drwx----- 3 root root 4096 Oct  6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct  6 2020 .local
-rw-r--r-- 1 root root 0 Feb 27 15:09 ls.txt
drwxr-xr-x 2 root root 4096 Feb 27 12:14 .nsight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct  6 2020 .selected_editor
drwx----- 2 root root 4096 Oct  6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw------- 1 root root 7455 Feb 22 14:53 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
root@linux-test:~#
```

표준 출력이 아닌 파일로 출력
(모니터로 출력 내용 없음)

출력이 저장된 파일 내용 확인

명령 >> 파일 : 명령의 표준 출력을 지정한 파일로 변경 (파일이 없으면 생성, 있으면 끝에 추가)
(정상 동작) (Append)

1. ls -ai 명령의 실행 결과를 ls.txt 파일 끝에 추가 : ls -ai >> ls.txt

```
root@linux-test:~# ls -ai >> ls.txt  
root@linux-test:~# cat ls.txt
```

명령 실행 결과를 파일의 끝에 추가
(모니터로 출력 내용 없음)

```
total 68  
drwx----- 9 root root 4096 Feb 27 15:30 .  
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..  
-rw----- 1 root root 68 Feb 27 11:19 .bash_history  
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout  
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc  
drwx----- 2 root root 4096 Feb 9 11:14 .cache  
drwx----- 3 root root 4096 Oct 6 2020 .gnupg  
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local  
-rw-r--r-- 1 root root 0 Feb 27 15:30 ls.txt  
drwxr-xr-x 2 root root 4096 Feb 27 12:14 .nsight  
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile  
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor  
drwx----- 2 root root 4096 Oct 6 2020 .ssh  
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim  
-rw----- 1 root root 7455 Feb 22 14:53 .viminfo  
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server  
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts  
524289 .  
2 ..  
524312 .bash_history  
524309 .bash_logout  
524290 .bashrc  
524292 .cache  
524296 .gnupg  
524305 .local  
524295 ls.txt  
524303 .nsight  
524291 .profile  
524301 .selected_editor  
524298 .ssh  
524300 .vim  
524310 .viminfo  
524313 .vscode-server  
524320 .wget-hsts
```

ls -ai 명령 결과 추가

에러 리다이렉션

명령 2> 파일 : 명령의 표준 에러 출력을 지정한 파일로 변경

stderr

1. ls 명령의 에러를 ls_err.txt 파일로 출력 : ls [없는 파일] 2> ls_err.txt

kkk는 없는 파일 (에러 발생)

에러 메시지를 ls_err.txt에 출력

```
root@linux-test:~# ls kkk 2> ls_err.txt
root@linux-test:~# cat ls_err.txt
ls: cannot access 'kkk': No such file or directory
root@linux-test:~#
```

파일에 저장된 에러 메시지

참고 : ls kkk를 출력 리다이렉션으로 파일에 저장할 경우

에러가 아닌 표준 출력을 리다이렉션

```
root@linux-test:~# ls kkk > ls_log.txt
ls: cannot access 'kkk': No such file or directory
root@linux-test:~# cat ls_log.txt
root@linux-test:~#
```

에러 메시지는 표준 에러(모니터)로 출력

파일 내용 없음

클력/에러 리다이렉션 응용

하나의 명령에서 정상 동작과 에러가 동시 발생

ls -al . + ls -al kkk
(정상 동작) (에러 발생)

```
root@linux-test:~# ls -al . kkk
ls: cannot access 'kkk': No such file or directory
:
total 68
drwx----- 9 root root 4096 Feb 28 10:40 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 28 09:14 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
drwxr-xr-x 2 root root 4096 Feb 28 09:12 .nsight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 8644 Feb 28 09:47 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
root@linux-test:~#
```

에러 출력

정상 출력

출력/에러 리다이렉션 응용

1. 표준 출력과 에러를 한번에 리다이렉션 : `ls -al . kkk > ls_log2 2> ls_err2`

```

root@linux-test:~# ls -al . kkk > ls_log2 2> ls_err2
root@linux-test:~#
root@linux-test:~# cat ls_log2
.:
total 72
drwx----- 9 root root 4096 Feb 28 10:53 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 28 09:14 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
-rw-r--r-- 1 root root 51 Feb 28 10:53 ls_err2
-rw-r--r-- 1 root root 0 Feb 28 10:53 ls_log2
drwxr-xr-x 2 root root 4096 Feb 28 09:12 .nsight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 8644 Feb 28 09:47 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
root@Linux-test:~#
root@Linux-test:~# cat ls_err2
ls: cannot access 'kkk': No such file or directory
root@Linux-test:~#

```

`ls -al . kkk` 명령에 대해

표준 출력은 `ls_log2` 파일로 리다이렉션

표준 에러는 `ls_err2` 파일로 리다이렉션

출력/에러 리다이렉션 응용

2. 표준 에러 출력 버리기 : ls -al . kkk 2> /dev/null

```
root@linux-test:~# ls -al . kkk 2> /dev/null
.:
total 76
drwx----- 9 root root 4096 Feb 28 10:53 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 28 09:14 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
-rw-r--r-- 1 root root 51 Feb 28 10:53 ls_err2
-rw-r--r-- 1 root root 924 Feb 28 10:53 ls_log2
drwxr-xr-x 2 root root 4096 Feb 28 09:12 .nfs01
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 8644 Feb 28 09:47 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
root@linux-test:~#
```

ls -al . kkk 명령에 대해

표준 에러를 /dev/null 파일로 리다이렉션

/dev/null (가상의 문자 장치 파일) :
출력 / 에러 내용을 버리는데 사용(블랙홀)

표준 출력만 모니터로 나타남

출력/에러 리다이렉션 응용

3. 표준 출력과 에러를 하나의 파일로 리다이렉션 : **ls -al . kkk > ls_log3 2>&1**

```
root@linux-test:~# ls -al . kkk > ls_log3 2>&1
root@linux-test:~# cat ls_log3
ls: cannot access 'kkk': No such file or directory
..
total 72
drwx----- 9 root root 4096 Feb 28 18:38 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 28 09:14 .bash_h
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_l
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
-rw-r--r-- 1 root root 51 Feb 28 18:38 ls_log3
drwxr-xr-x 2 root root 4096 Feb 28 12:42 .nsight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 8479 Feb 28 18:26 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
root@linux-test:~#
```

표준 출력은 ls_log3 파일로 리다이렉션

표준에러는 변경된 표준 출력(ls_log3)로 리다이렉션

&1 : 표준 출력의 파일 디스크립터

2>와 &1는 붙여야 한다 (공백X)

표준 출력만 나타남

표준 출력/에러 모두 파일에 기록

입력 리다이렉션

명령 < 파일 : 명령의 표준 입력을 지정한 파일로 변경 (파일로부터 입력)

(명령 0< 파일)

stdin

1. .bashrc 파일의 내용을 입력받아 출력 : **cat < .bashrc**
cat 0< .bashrc

cat .bashrc와 동일

(< 가 생략된 형태라고 볼 수 있다)
(head, tail, wc 등도 마찬가지)

```
root@linux-test:~# cat < .bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
[ -z "$PS1" ] && return

# don't put duplicate lines in the history. See bash(1) for more options
# ... or force ignoreups and ignorespace
HISTCONTROL=ignoreups:ignorespace

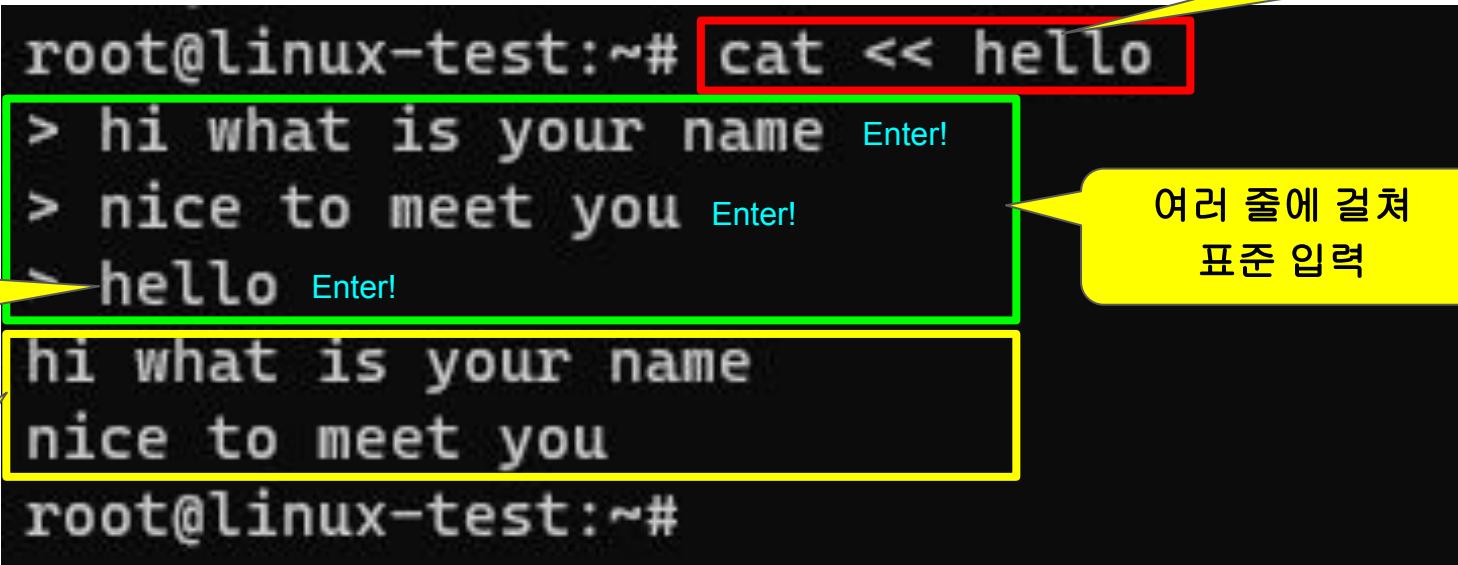
# append to the history file, don't overwrite it
shopt -s histappend
```

파일의 내용을 먼저
입력받고 출력

입력 리다이렉션

명령 << 구분문자열 : << 뒤에 지정한 구분문자열이 다시 입력될 때까지 표준 입력을 수행
이후 구분문자열 이전까지의 내용에 대해 명령 수행 (**Heredoc 구문**)

1. << 리다이렉션을 이용한 cat 명령 실행



The screenshot shows a terminal window with the following interaction:

```
root@linux-test:~# cat << hello
> hi what is your name Enter!
> nice to meet you Enter!
> hello Enter!
```

Annotations explain the components:

- A yellow callout points to the text "hi what is your name" with the text "구분 문자열 다시 입력".
- A yellow callout points to the text "hello" with the text "여러 줄에 걸쳐 표준 입력".
- A yellow callout points to the entire command line with the text "cat 명령 수행 (표준 출력)".
- A yellow callout points to the label "구분 문자열 (hello)" above the terminal window.

The terminal output shows the standard output of the cat command, which is the input provided between the << redirection operator and the end of file marker (EOF).

입력/출력 리다이렉션 응용

- 표준 입력으로 받은 내용을 hello 파일에 출력(저장)

```
root@linux-test:~# cat << EOF > hello Enter!
> Hi, How are you Enter!
> I'm fine Enter!
> Thank you Enter!
> EOF Enter!
root@linux-test:~# cat hello
```

표준 입력

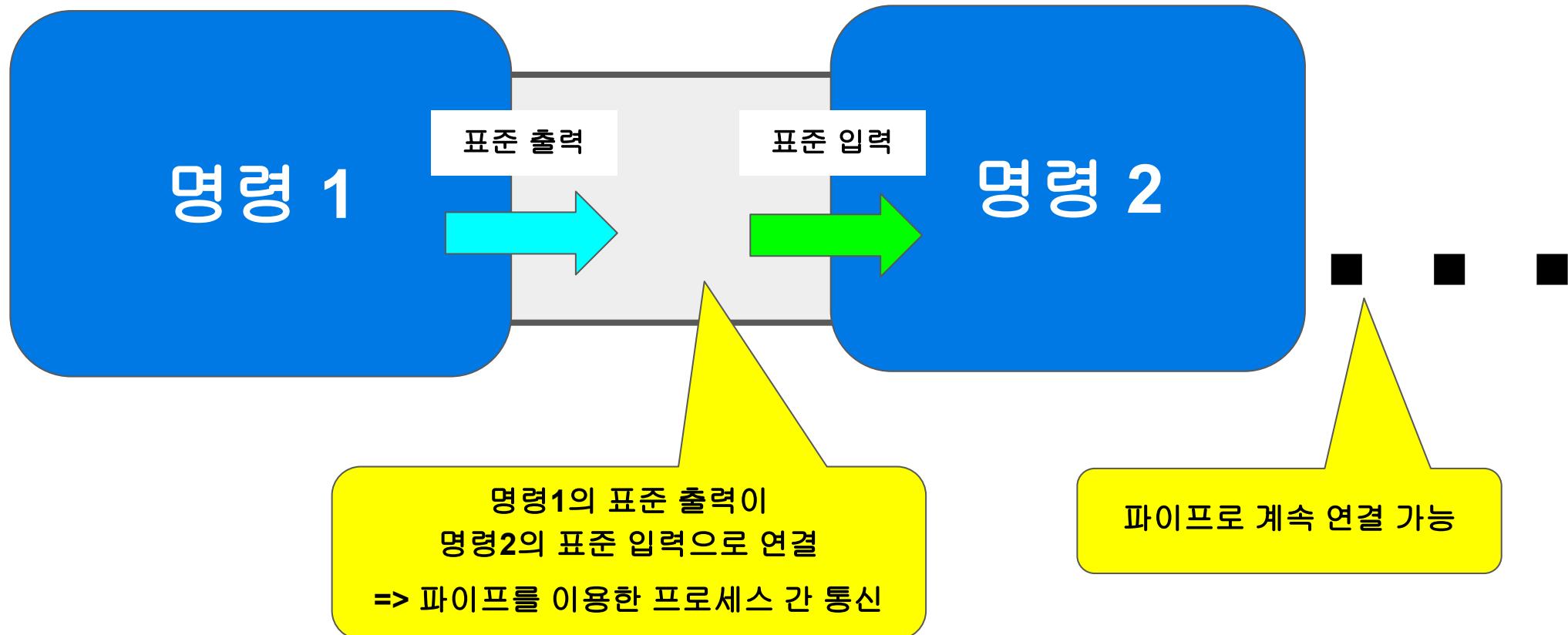
cat << EOF > hello

cat << EOF : 다음 EOF 이전까지 (표준)입력
받은 내용을 (표준) 출력

> hello : 표준 출력을 hello 파일로 변경

hello 파일에 저장된 내용

파이프(Pipe)



파이프(Pipe)

명령1 | 명령2 : 명령1의 표준 출력을 명령2의 표준 입력으로 연결

1. ls -al 의 출력 중에서 앞부분 5줄 출력 : ls -al | **head -5**

명령에 파일명 인자가 없을 경우 표준 입력이 대신한다
(cat, wc, head, tail 등)

=> ls -al 명령의 출력이 head 명령의 파일 인자를 대신한다.

ls -al

```
total 68
drwx----- 9 root root 4096 Feb 28 22:32 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 28 09:14 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
drwxr-xr-x 2 root root 4096 Feb 28 12:42 .insight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 8479 Feb 28 18:26 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
```

표준 출력

head -5

```
total 68
drwx----- 9 root root 4096 Feb 28 22:32 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root 0 Feb 28 09:14 .bash_history
-rw-r--r-- 1 root root 33 Feb 22 14:53 .bash_logout
-rw-r--r-- 1 root root 3106 Apr 9 2018 .bashrc
drwx----- 2 root root 4096 Feb 9 11:14 .cache
drwx----- 3 root root 4096 Oct 6 2020 .gnupg
drwxr-xr-x 3 root root 4096 Oct 6 2020 .local
drwxr-xr-x 2 root root 4096 Feb 28 12:42 .insight
-rw-r--r-- 1 root root 148 Aug 18 2015 .profile
-rw-r--r-- 1 root root 75 Oct 6 2020 .selected_editor
drwx----- 2 root root 4096 Oct 6 2020 .ssh
drwxr-xr-x 2 root root 4096 Oct 12 2020 .vim
-rw----- 1 root root 8479 Feb 28 18:26 .viminfo
drwxr-xr-x 5 root root 4096 Feb 20 12:16 .vscode-server
-rw-r--r-- 1 root root 183 Feb 20 12:16 .wget-hsts
```

표준 입력

ls -al 출력의 앞부분 5줄 출력

파이프(Pipe)

ls -al | head -5 실행

```
root@linux-test:~# ls -al | head -5
total 68
drwx----- 9 root root 4096 Feb 28 22:32 .
drwxr-xr-x 25 root root 4096 Oct 13 2020 ..
-rw----- 1 root root     0 Feb 28 09:14 .bash_history
-rw-r--r-- 1 root root   33 Feb 22 14:53 .bash_logout
root@linux-test:~#
```

파이프 응용

sed 명령과의 조합

표준 입력(또는 파일)으로 받은 내용을 편집하여 출력

(자세한 사용법은 필요할 때 직접 찾아보세요~)

(지금은 이토록 다양하게 쓸 수 있다는 것만 알고 갑시다^^)

- 확장자가 .txt인 파일의 검색 리스트를 출력하고, 출력에서 확장을 .text로 변경한 후, 변경 내용을 text2 파일에 저장 :

```
find / -name “*.txt” | sed ‘s/\.\txt/\.\text/g’ > text2
```

파일 검색

출력 내용(파일 검색)을 편집

편집 내용을
파일로 저장

```
root@linux-test:~# find / -name “*.txt” | sed ‘s/\.\txt/\.\text/g’ > text2
```

```
root@linux-test:~# head -5 text2
/lib/firmware/ath10k/QCA4019/hw1.0/notice_ath10k_firmware-5.text
/lib/firmware/ath10k/QCA9888/hw2.0/notice_ath10k_firmware-5.text
/lib/firmware/ath10k/QCA6174/hw3.0/notice_ath10k_firmware-6.text
/lib/firmware/ath10k/QCA6174/hw3.0/notice_ath10k_firmware-4.text
/lib/firmware/ath10k/QCA6174/hw2.1/notice_ath10k_firmware-5.text
root@linux-test:~#
```

문자열 검색
및 교체 패턴

awk 명령과의 조합

표준 입력(또는 파일)으로 받은 내용 중 필요한 정보를
추출하거나 원하는 형식으로 출력
(자세한 사용법은 직접 찾아보세요~)

- ls -al 명령에서 파일 이름과 권한 정보만 출력 : ls -al | awk 'NR>1{print \$1, \$9}'

NR>1 : 행(record) 번호가
1보다 큰 영역

ls -al

{print \$1, \$9} : 번호 1, 9번 열(field)
의 데이터 출력
(각 열은 공백으로 구분)

awk 'NR>1{print \$1, \$9}'

		\$1	\$2	\$3	\$4	\$5	\$6	\$7	\$8	\$9
total	104									
drwx-----	9	root	root	4096	Mar 2	10:38	.			
drwxr-xr-x	25	root	root	4096	Oct 13	2020	..			
-rw-----	1	root	root	0	Mar 2	09:30	.bash_history			
-rw-r--r--	1	root	root	33	Feb 22	14:53	.bash_logout			
-rw-r--r--	1	root	root	3106	Apr 9	2018	.bashrc			
drwx-----	2	root	root	4096	Feb 9	11:14	.cache			
drwx-----	3	root	root	4096	Oct 6	2020	.gnupg			
drwxr-xr-x	3	root	root	4096	Oct 6	2020	.local			
drwxr-xr-x	2	root	root	4096	Mar 2	09:26	.nsight			
-rw-r--r--	1	root	root	148	Aug 18	2015	.profile			
-rw-r--r--	1	root	root	75	Oct 6	2020	.selected_editor			
drwx-----	2	root	root	4096	Oct 6	2020	.ssh			
-rw-r--r--	1	root	root	33954	Mar 2	10:39	text2			
drwxr-xr-x	2	root	root	4096	Oct 12	2020	.vim			
-rw-----	1	root	root	8586	Mar 2	10:38	.viminfo			
drwxr-xr-x	5	root	root	4096	Feb 20	12:16	.vscode-server			
-rw-r--r--	1	root	root	183	Feb 20	12:16	.wget-hsts			

파이프 응용

ls -al | awk 'NR>1{print \$1, \$9}' 실행

```
root@linux-test:~# ls -al | awk 'NR>1{print $1, $9}'  
drwx----- .  
drwxr-xr-x ..  
-rw----- .bash_history  
-rw-r--r-- .bash_logout  
-rw-r--r-- .bashrc  
drwx----- .cache  
drwx----- .gnupg  
drwxr-xr-x .local  
drwxr-xr-x .nsight  
-rw-r--r-- .profile  
-rw-r--r-- .selected_editor  
drwx----- .ssh  
-rw-r--r-- text2  
drwxr-xr-x .vim  
-rw----- .viminfo  
drwxr-xr-x .vscode-server  
-rw-r--r-- .wget-hsts  
root@linux-test:~#
```

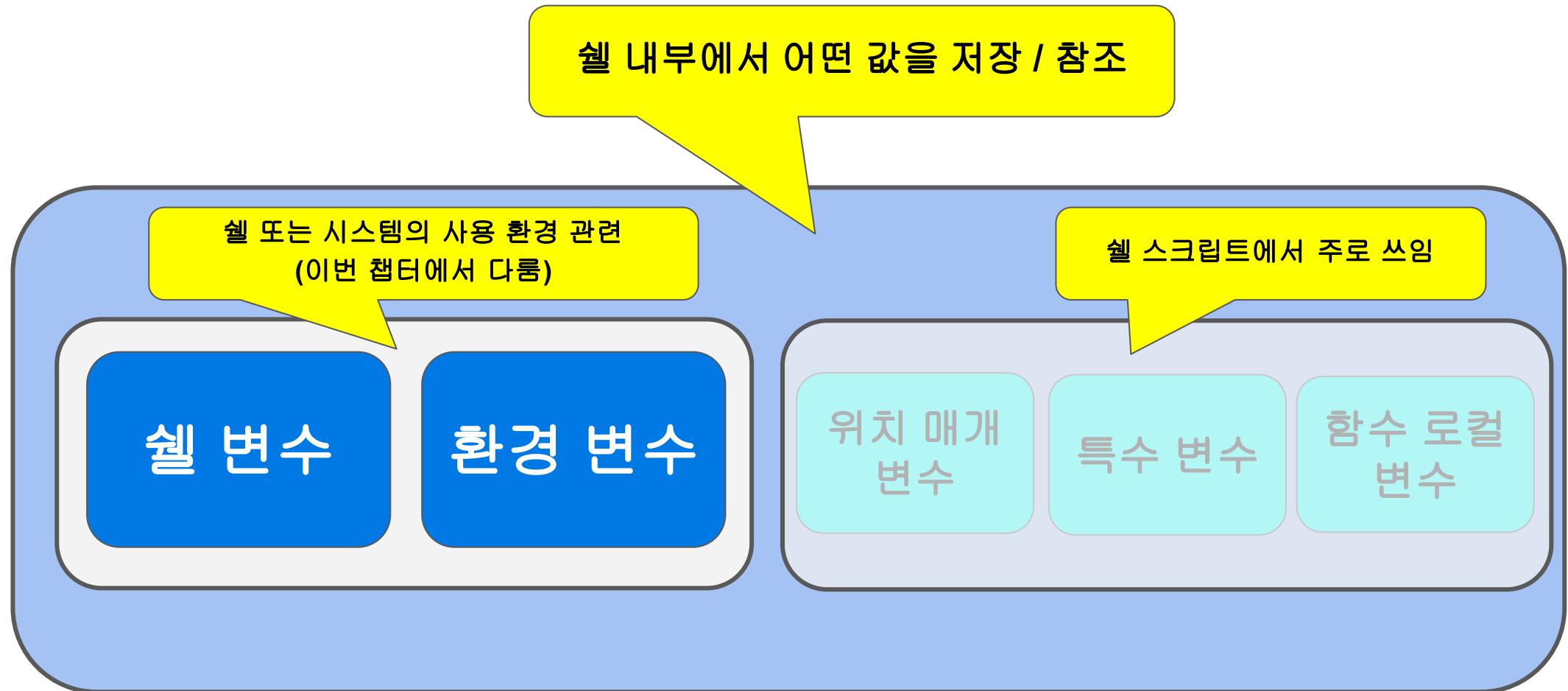
명령의 선택 + 조합

상황에 맞는 명령을 선택하고 조합하는 능력

=> 어떤 상황에서 어떤 명령을 어떻게 조합해 쓸 것인가?

=> 경험이 필요 (ChatGPT 활용해 보세요!)

Shell과 변수



쉘 변수

VS

환경 변수

변수를 선언한 쉘에서만 참조 가능

변수의 유효 범위에서 차이

변수를 선언한 쉘 (1)

+

(1)에서 실행된 프로그램
에서 참조 가능

변수의 출력

set: 쉘 변수 + 환경변수 + 쉘 함수를 출력

```
root@linux-test:~# set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force_fignore:histappend:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSINFO=( [0]=""2" [1]=""8")
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=( [0]=""4" [1]=""4" [2]=""20" [3]=""1" [4]=""release" [5]=""x86_64-pc-linux-gnu")
BASH_VERSION='4.4.20(1)-release'
COLUMN=109
```

출력되는 내용이
상당히 많다

(중략)

```
{
    if [[ ${!1} == \~* ]]; then
        eval $1=$(printf ~%q "${!1#\~}");
    fi
}
_get_cword_at_cursor_by_ref ()
{
    local cword words=();
    _reassemble_comp_words_by_ref "$1" words cword;
    local i cur index=${COMP_POINT} lead=${COMP_LINE:0:${COMP_POINT}};
    if [[ $index -gt 0 && ( -n $lead && -n ${lead//[:space:]} ) ]]; then
        cur=${COMP_LINE};
        for ((i = 0; i <= cword; ++i))
        do
            while [[ ${#cur} -ge ${#words[i]} && "${cur:0:${#words[i]}}" != "${words[i]}" ]]; do
                cur="${cur:1}";
                [[ $index -gt 0 ]] && ((index--));
            done;
            if [[ $i -lt $cword ]]; then
                local old_size=${#cur};
                cur="${cur}${words[i]}";
                local new_size=${#cur};
                index=$(( index - old_size + new_size ));
            fi;
        done;
        [[ -n $cur && ! -n ${cur//[:space:]} ]] && cur=;
        [[ $index -lt 0 ]] && index=0;
    fi;
    local "$2" "$3" "$4" && _upvars -a${#words[@]} $2 "${words[@]}" -v $3 "$cword" -v $4 "${cur:0:$index}"
}
```

로그인 쉘(bash)의 주요 쉘 변수

HISTSIZE	명령어 히스토리 스택에 저장 가능한 명령 개수
PS1	프롬프트 문자열 정의
IFS	필드 구분자 (공백, 탭, 개행)

변수의 출력

env : 환경 변수만 출력

변수 검색 시 **set**, **env** 명령과 **grep**과의
파이프 조합으로 수행

```
root@linux-test:~# env
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi
=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*
.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01
;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31
:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01
;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.7z
o=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31
.jpg=01;35:*.jpeg=01;35:*.jpg=01;35:*.mpeg=01;35:*.gif=01;35:*.bmp=01;35:*
=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01
:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*
=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01
.rmvb=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl
:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*
0;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;
av=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36
SSH_CONNECTION=221.146.63.213 22643 10.41.134.235 22
LESSCLOSE=/usr/bin/lesspipe %s %s
LANG=en_US.UTF-8
S_COLORS=auto
XDG_SESSION_ID=69
USER=root
PWD=/root
HOME=/root
SSH_CLIENT=221.146.63.213 22643 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
SSH_TTY=/dev/pts/0
MAIL=/var/mail/root
TERM=xterm-256color
SHELL=/bin/bash
SHLVL=1
LANGUAGE=en_US:en
LOGNAME=root
XDG_RUNTIME_DIR=/run/user/0
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
LESSOPEN=-| /usr/bin/lesspipe %s
```

주요 환경 변수

HOME	사용자 홈 디렉토리의 절대 경로
USER	사용자 이름
LANG	사용하는 언어 (영어)
LOGNAME	사용자 계정 이름
PATH	명령 탐색 경로
PWD	현재 작업 디렉토리의 절대 경로
SHELL	로그인 쉘

변수의 출력

\$: 뒤에 오는 문자열을 변수로 취급 (특수문자)

echo \$변수명 : 개별 변수의 값을 출력

```
root@linux-test:~# echo $SHELL  
/bin/bash
```

SHELL 변수의 값을 출력

쉘 변수의 선언(설정)

변수명=**값**(문자열)

변수에 저장되는 **모든 값은 문자열** (숫자도 문자열로 취급)

변수 이름은 대소문자 구분함

변수 대입할 때 = 의 좌우에는 **공백 없어야 함**

```
root@linux-test:~# THIS_CLASS=linux
root@linux-test:~# echo $THIS_CLASS
linux
root@linux-test:~# set | grep THIS_CLASS
THIS_CLASS=linux
root@linux-test:~# env | grep THIS_CLASS
root@linux-test:~#
```

THIS_CLASS라는 이름의
쉘 변수 값을 linux로 설정

환경변수가 아니므로 env 출력에서는
검색되지 않음

환경 변수의 선언

● export

- **기능** : 이전에 선언한 쉘 변수를 환경변수로 변경
- **형식** : **export [옵션] [쉘 변수]**

1. 쉘 변수 THIS_CLASS를 환경변수로 변경 : **export THIS_CLASS**

기존에 쉘 변수로 선언된
THIS_CLASS를 환경변수로 선언
=> env 출력에서 검색됨

```
root@linux-test:~# export THIS_CLASS
root@linux-test:~# env | grep THIS_CLASS
THIS_CLASS=linux
```

한번에 환경변수로 선언

2. OTHER_CLASS를 환경변수로 선언하고 값을 java로 대입 : **export THIS_CLASS=java**

```
root@linux-test:~# export OTHER_CLASS=java
root@linux-test:~# env | grep OTHER_CLASS
OTHER_CLASS=java
```

환경 변수의 선언

- **export**

3. 환경변수 THIS_CLASS를 쉘 변수로 변경 : **export -n THIS_CLASS**

```
root@linux-test:~# env | grep THIS_CLASS  
THIS_CLASS=linux  
root@linux-test:~# export -n THIS_CLASS  
root@linux-test:~#  
root@linux-test:~# env | grep THIS_CLASS  
root@linux-test:~# set | grep THIS_CLASS  
THIS_CLASS=linux
```

환경변수를 쉘 변수로 변경

환경변수에서 검색 안됨

전체 변수에서 검색됨

변수 선언의 다른 방법

- **declare**

쉘 변수 : **declare** 변수명=값

환경 변수 : **declare -x** 변수명=값

-x : export의 의미

```
root@linux-test:~# declare MY_VAR=hello
root@linux-test:~# set | grep MY_VAR
MY_VAR=hello
_=MY_VAR=hello
root@linux-test:~# env | grep MY_VAR
root@linux-test:~#
root@linux-test:~#
root@linux-test:~# declare -x MY_VAR2=world
root@linux-test:~# env | grep MY_VAR2
MY_VAR2=world
```

쉘 변수 선언

환경 변수 선언

변수의 해제

● unset

- **기능** : 지정한 변수를 해제 (쉘 변수, 환경변수 모두 해당)
- **형식** : **unset [변수]**

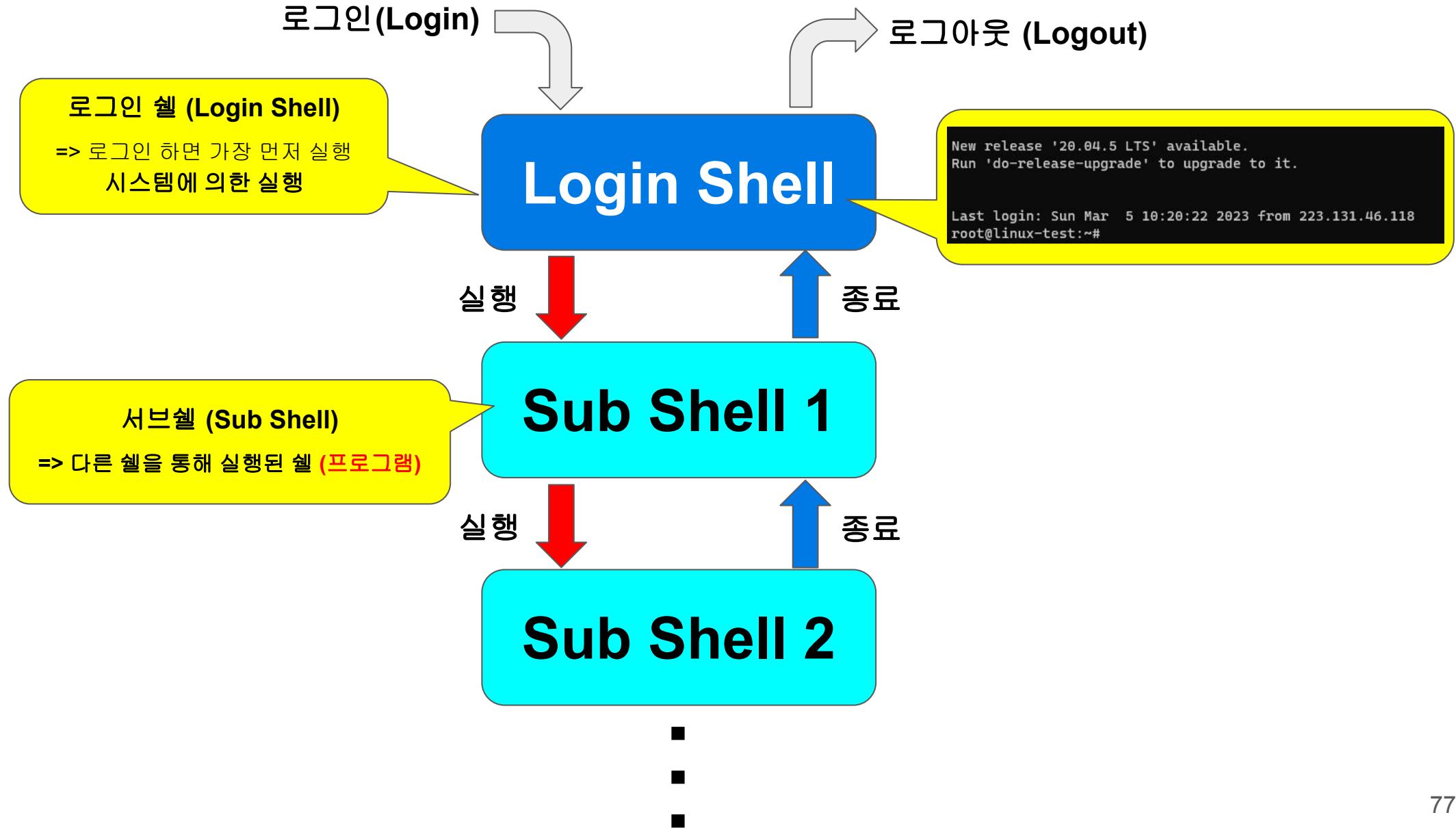
1. 환경변수 OTHER_CLASS를 해제 : **unset OTHER_CLASS**

```
root@linux-test:~# env | grep OTHER_CLASS
OTHER_CLASS=java
root@linux-test:~# unset OTHER_CLASS
root@linux-test:~# env | grep OTHER_CLASS
root@linux-test:~# set | grep OTHER_CLASS
root@linux-test:~#
```

환경 변수로 선언되어 있던
OTHER_CLASS를 해제

변수 목록에서 검색되지 않음

변수의 유효 범위 - 로그인 쉘 / 서브쉘



변수의 유효 범위 -로그인 쉘 / 서브쉘

로그인 쉘 (bash) 사용 구간

```
New release '20.04.5 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.
```

```
Last login: Thu Mar  2 15:35:55 2023 from 221.146.63.213
```

```
root@linux-test:~# ls -a
```

```
.
..
a.out
.bash_history
root@linux-test:~#
root@linux-test:~# sh
```

sh(dash)라는 서브쉘 실행

=> 프롬프트 모양이 변경

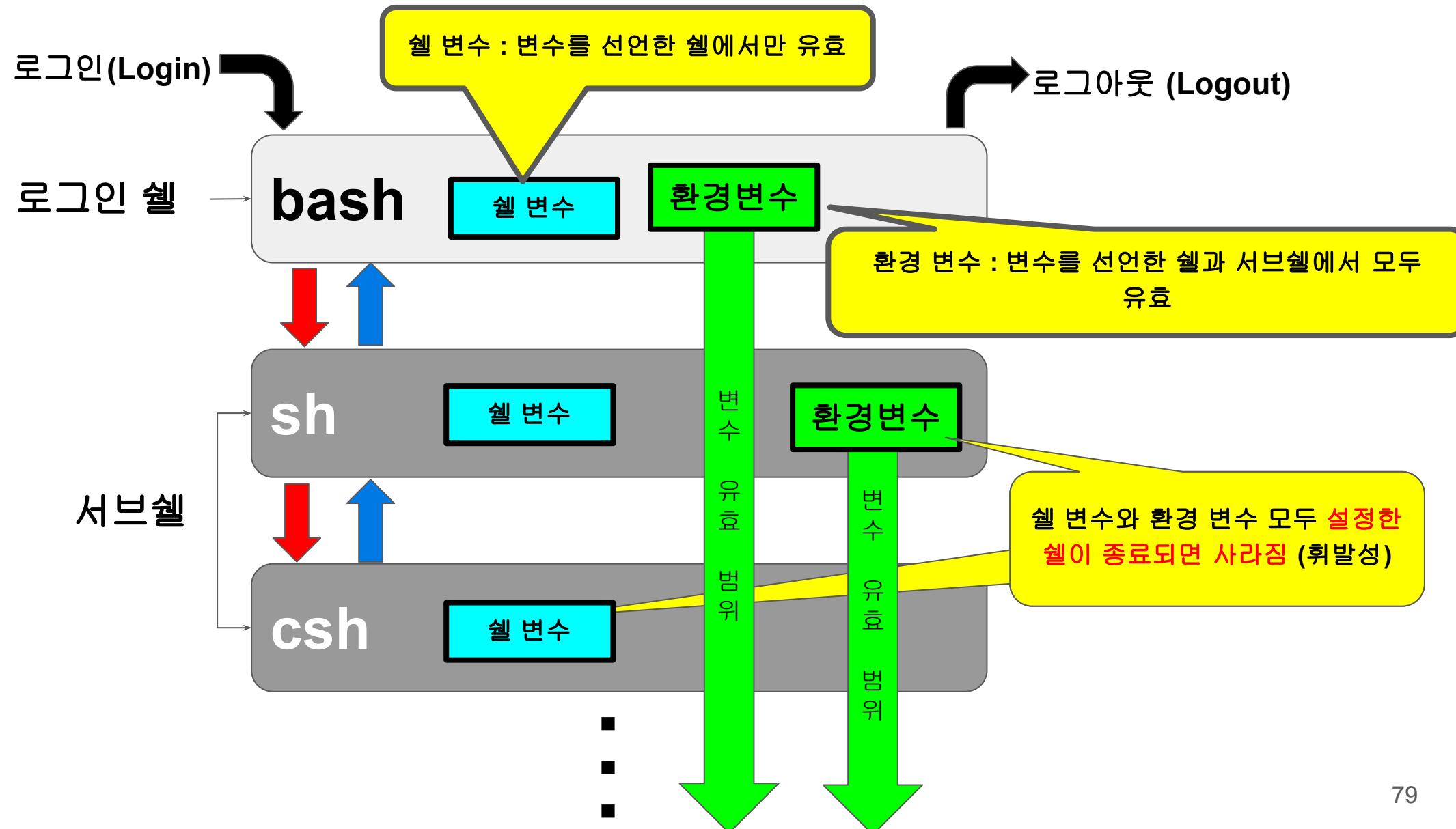
```
# ls -a
```

```
.
..
a.out
.bash_history
#
# exit
root@linux-test:~#
```

로그인 쉘(bash)로 복귀

서브쉘(sh) 사용 구간

변수의 유효 범위



변수의 유효 범위

쉘 변수

```
root@linux-test:~# MY_VAR=hello  
root@linux-test:~# set | grep MY_VAR  
MY_VAR=hello  
root@linux-test:~# sh  
# set | grep MY_VAR  
#  
# exit  
root@linux-test:~# set | grep MY_VAR  
MY_VAR=hello  
root@linux-test:~#
```

쉘 변수는 서브쉘에서
검색되지 않음

쉘 변수는 그 변수를 선언한
쉘에서만 유효

환경 변수는 서브쉘에서도 검색

환경 변수

```
root@linux-test:~# export MY_VAR2=world  
root@linux-test:~# env | grep MY_VAR2  
MY_VAR2=world  
root@linux-test:~# sh  
# set | grep MY_VAR2  
MY_VAR2='world'  
# env | grep MY_VAR2  
MY_VAR2=world  
# exit  
root@linux-test:~# env | grep MY_VAR2  
MY_VAR2=world  
root@linux-test:~#
```

변수의 유효 범위

서브쉘에서의 환경 변수 선언

```
root@linux-test:~#  
root@linux-test:~# sh  
# export MY_VAR3=hahaha  
# env | grep MY_VAR3  
_=MY_VAR3=hahaha  
MY_VAR3=hahaha  
#  
# sh  
# env | grep MY_VAR3  
_=MY_VAR3=hahaha  
MY_VAR3=hahaha  
# exit  
#  
# env | grep MY_VAR3  
MY_VAR3=hahaha  
# exit  
root@linux-test:~# env | grep MY_VAR3  
root@linux-test:~#  
root@linux-test:~# sh  
# env | grep MY_VAR3  
#
```

서브쉘1 범위

서브쉘1에서 환경 변수 선언

서브쉘2 범위

로그인 쉘에서는 서브쉘에서 선언한
환경변수가 검색되지 않음
=> 선언한 쉘 종료 시 해제

변수의 유효 범위

서브쉘에서의 환경 변수 변경

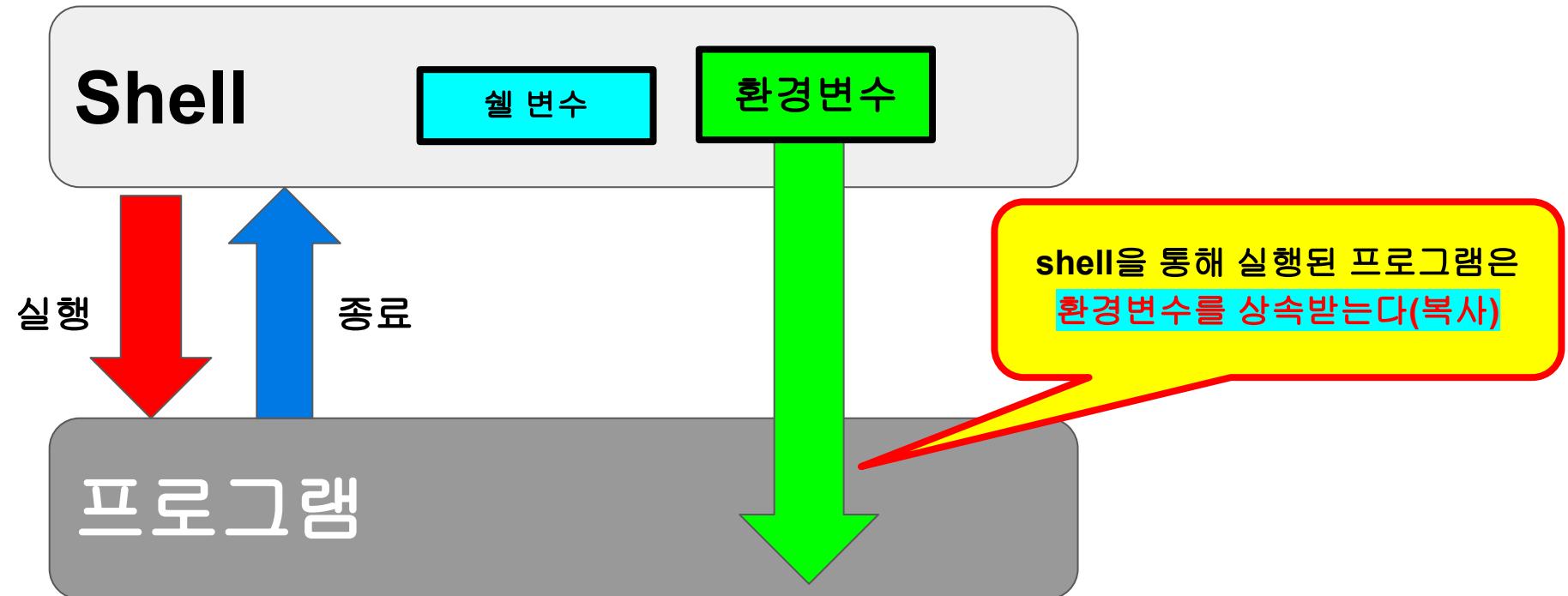
```
root@linux-test:~# export MY_VAR4=hello  
root@linux-test:~# sh  
# env | grep MY_VAR4  
MY_VAR4=hello  
# MY_VAR4=world  
# env | grep MY_VAR4  
MY_VAR4=world  
# exit  
root@linux-test:~# env | grep MY_VAR4  
MY_VAR4=hello  
root@linux-test:~#
```

로그인 쉘에서 환경 변수 설정

로그인 쉘에서 선언한 환경 변수 값을
서브쉘에서 변경

서브쉘에서 변경한 값이
로그인 쉘에는 적용되지 않음

환경 변수의 상속



환경 변수의 상속 - node.js

```
root@linux-test:~# node
> process.env
{
  LS_COLORS: 'rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;3
  0;33:01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=3
  4;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.l
  zh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.tz=01;3
  1:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.l
  z=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.b
  z=01;31:*.tbz=01;31:*.tbz2=01;31:*.taz=01;31:*.deb=01;31:*.rpm=01;31:*.j
  ar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.a
  ce=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.w
  im=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mj
  pg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=0
  1;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=
  01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=
  01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=
  01;35:*.nuv=01;35:*.wmv=01;35:*.ASF=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=
  01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;3
  5:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4
  a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.w
  av=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:*
  SSH_CONNECTION: '223.131.46.118 36078 10.41.134.235 22',
  LESSCLOSE: '/usr/bin/lesspipe %s %s',
  LANG: 'en_US.UTF-8',
  OLDPWD: '/',
  S_COLORS: 'auto',
  XDG_SESSION_ID: '59',
  USER: 'root',
  PWD: '/root',
  HOME: '/root',
  SSH_CLIENT: '223.131.46.118 36078 22',
  XDG_DATA_DIRS: '/usr/local/share:/usr/share:/var/lib/snapd/desktop',
  SSH_TTY: '/dev/pts/1',
  MAIL: '/var/mail/root',
  TERM: 'xterm-256color',
  SHELL: '/bin/bash',
  SHLVL: '1',
  LANGUAGE: 'en_US:en',
  LOGNAME: 'root',
  XDG_RUNTIME_DIR: '/run/user/0',
  PATH: '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin',
  LESSOPEN: '| /usr/bin/lesspipe %s',
  -> '/usr/bin/node'
}
>
```

shell에서 node.js 실행 (REPL)

process.env 객체로
환경변수 조회 가능!!

환경 변수의 상속 - node.js

Shell (환경변수 선언)

```
# API_KEY와 PASSWORD 환경 변수를 설정합니다.  
export API_KEY="your_api_key"  
export PASSWORD="your_password"
```

API_KEY, PASSWORD와 같은 민감한
정보를 소스코드와 분리 가능!!
보안성, 유연성 강화

node.js 기반 코드 (Server)

```
const apiKey = process.env.API_KEY;  
const password = process.env.PASSWORD;  
  
// apiKey와 password를 사용하여 API 요청을 보냅니다.  
sendApiRequest(apiKey, password);
```

변수의 산술 연산

● expr (**expression** : 표현식)

- **기능** : 쉘에서 산술, 비교 등의 연산 실행
- **형식** : **expr [EXPRESSION]**

EXPRESSION은 산술, 비교 연산식 등에 해당

연산식에서 값과 연산자 사이는
반드시 공백 한칸 필요

숫자 대신 변수를 이용한 연산도 가능

1. 산술 연산 예시

```
root@linux-test:~# expr 5 + 4
9
root@linux-test:~# expr 5 - 4
1
root@linux-test:~# expr 5 \* 4
20
root@linux-test:~# expr 5 / 4
1
root@linux-test:~# expr 5 % 4
1
```

쉘에서 입력되는 모든 값은 **문자**
산술 연산을 위해서는 **expr**과 같은 도구 필요
expr 뒤에 붙는 내용을 **수식으로 취급**

변수를 이용한 산술 연산

```
root@linux-test:~# VAR1=5
root@linux-test:~# VAR2=4
root@linux-test:~# expr $VAR1 + $VAR2
9
root@linux-test:~# expr $VAR1 - $VAR2
1
root@linux-test:~# expr $VAR1 \* $VAR2
20
root@linux-test:~# expr $VAR1 / $VAR2
1
root@linux-test:~# expr $VAR1 % $VAR2
1
```

변수의 산술 연산

- **\$(())**, **\$[]**

- **기능** : 쉘에서 산술, 비교 등의 연산 실행
- **형식** : **\$((EXPRESSION))**, **\$[EXPRESSION]**

값과 연산자 사이에 **공백 필요 없음**

```
root@linux-test:~# VAR1=3
root@linux-test:~# VAR2=2
root@linux-test:~# echo $((VAR1+VAR2))
5
root@linux-test:~# echo ${VAR1-$VAR2}
1
root@linux-test:~# echo $((VAR1*$VAR2))
6
root@linux-test:~# echo ${VAR1*$VAR2}
6
```

곱셈 연산 시에도 '**'** 문자 필요 없음

Shell 스크립트 프로그래밍

쉘 프롬프트에서 실행 가능한 모든 명령을

주어진 문법에 맞게

원하는 순서로

쉘을 통해 실행하도록 작성한

(실행 가능한) 텍스트 파일

쉘 스크립트 기본 형식

greeting.sh

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

주석 : #으로 시작

shebang(셔뱅) : #!로 시작하여, 아래의
스크립트를 실행할 쉘의 경로 지정

```
# 변수 선언 및 할당
```

```
name="Alice"
```

```
age=25
```

실행 명령

```
# 함수 선언
```

```
function greet {
```

```
    echo "안녕하세요, $name($age)님!"
```

```
}
```

```
# 함수 호출
```

```
greet
```

```
# 조건문
```

```
if [ $age -lt 18 ]
```

```
then
```

```
    echo "미성년자입니다."
```

```
elif [ $age -lt 65 ]
```

```
then
```

```
    echo "성인입니다."
```

```
else
```

```
    echo "노인입니다."
```

```
fi
```

쉘 스크립트 문법

쉘 스크립트 실행

1. 프롬프트에서 실행할 쉘을 명시하고 스크립트 파일을 인자로 전달

```
root@linux-test:~# bash greeting.sh  
안녕하세요, Alice(25) 님!  
성인입니다.
```

bash 명령으로 greeting.sh 스크립트를 실행

2. 프롬프트에서 스크립트 파일을 직접 실행

파일에 실행 권한 부여
`-rwxr-xr-x`

```
root@linux-test:~# chmod 755 greeting.sh  
root@linux-test:~# ls -al greeting.sh  
-rwxr-xr-x 1 root root 317 Mar  5 22:39 greeting.sh  
root@linux-test:~# ./greeting.sh  
안녕하세요, Alice(25) 님!  
성인입니다.  
root@linux-test:~#
```

shebang에 지정된 쉘로 스크립트 실행

쉘 스크립트 실행

3. 스크립트 실행 예러

```
root@linux-test:~# sh greeting.sh
greeting.sh: 8: greeting.sh: function: not found
안녕하세요, Alice(25) 님!
greeting.sh: 10: greeting.sh: Syntax error: ")" unexpected
root@linux-test:~#
```

bash 문법으로 작성된 파일을 sh(dash)로 실행 시 예러 발생
쉘마다 스크립트 문법이 약간씩 차이가 있으므로 주의

리눅스 시스템 관리의 자동화에 도움

리눅스 시스템 관리



파일 복사, 백업, 빌드 등

사용자 환경 구성, 부팅 스크립트 등

시스템 자원, 네트워크 등

쉘 스크립트가 부적합한 작업

- 빠른 속도가 필요한 작업 (정렬, 해쉬 등)
- 강력한 산술 연산 (실수, 복소수 등)
- 보안상 중요한 작업 등

쉘 스크립트 예제 - if 조건문

1. 숫자 비교하기

```
#!/bin/bash
age=20
if [ $age -lt 18 ]
then
    echo "You are too young"
elif [ $age -lt 25 ]
then
    echo "You are in the prime of your life"
else
    echo "You are getting old"
fi
```

변수 값에 대한 산술 비교
(변수 값을 숫자로 취급)

산술 비교	결과
수식1 -eq 수식2	두 수식(또는 변수)이 같으면 참
수식1 -ne 수식2	두 수식(또는 변수)이 같지 않으면 참
수식1 -gt 수식2	수식1이 크면 참
수식1 -ge 수식2	수식1이 크거나 같으면 참
수식1 -lt 수식2	수식1이 작으면 참
수식1 -le 수식2	수식1이 작거나 같으면 참
!수식	수식이 거짓이면 참

쉘 스크립트 예제 – if 조건문

2. 문자열 비교하기

```
#!/bin/bash
username="admin"
password="pass123"
if [ $username == "admin" ] && [ $password == "pass123" ]
then
    echo "Access granted"
else
    echo "Access denied"
fi
```

[조건1] && [조건2] : 조건1과 조건2가 모두 참이면 참 (AND 연산)
[조건1] || [조건2] : 조건1, 조건2 둘 중 하나가 참이면 참 (OR 연산)

변수 값에 대한 문자열 비교

문자열 비교	결과
“문자열1”=“문자열2”	두 문자열이 같으면 참
“문자열1”!=“문자열2”	두 문자열이 다르면 참
-n “문자열”	문자열이 NULL(빈 문자열)이 아니면 참
-z “문자열”	문자열이 NULL(빈 문자열)이면 참

쉘 스크립트 예제 – if 조건문

3. 파일 존재 여부 확인하기

```
#!/bin/bash
filename="myfile.txt"
if [ -e $filename ]
then
    echo "$filename exists"
else
    echo "$filename does not exist"
fi
```

파일 관련 조건

파일 조건	결과
-d 파일이름	파일이 디렉토리이면 참
-e 파일이름	파일이 존재하면 참
-f 파일이름	파일이 일반 파일이면 참
-r 파일이름	파일이 읽기 가능하면 참
-w 파일이름	파일이 쓰기 가능하면 참
-x 파일이름	파일이 실행 가능하면 참
-u 파일이름	파일에 set-user-id가 설정되면 참
-g 파일이름	파일에 set-group-id가 설정되면 참
-s 파일이름	파일 크기가 0이 아니면 참

쉘 스크립트 예제 – case 조건문

1. case 조건 예제

```
#!/bin/bash
echo "What is your favorite color?"
```

```
read color
```

표준 입력으로 받아 color 변수에 저장

```
case $color in
```

```
"red")
```

```
    echo "You like red"
```

```
;;
```

```
"green")
```

```
    echo "You like green"
```

```
;;
```

```
"blue")
```

```
    echo "You like blue"
```

```
;;
```

```
*)
```

```
    echo "You did not enter a valid color"
```

```
;;
```

```
esac
```

축약

```
#!/bin/bash
echo "What is your favorite color?"
read color
```

```
case $color in
```

```
"red"|"green"|"blue")
```

```
    echo "You like $color"
```

```
;;
```

```
*)
```

```
    echo "You did not enter a valid color"
```

```
;;
```

```
esac
```

color 값이 “red”, “green”, “blue” 중
하나면 실행

*) : 모든 조건에 맞지 않는 경우 (else)

쉘 스크립트 예제 - 반복문

1. for ~ in

apple, banana, orange가 순서대로
fruit에 저장되고 반복

```
#!/bin/bash
for fruit in apple banana orange
do
    echo "I like $fruit"
done
```

3. until

조건식이 거짓인 동안 반복

```
#!/bin/bash
count=1
until [ $count -gt 5 ]
do
    echo "Count is $count"
    count=$((count+1))
done
```

2. while

조건식이 참인 동안 반복

```
#!/bin/bash
count=1
while [ $count -le 5 ]
do
    echo "Count is $count"
    count=$((count+1))
done
```

4. break (반복문 종료)

```
#!/bin/bash
for fruit in apple banana orange
do
    if [ $fruit = "banana" ]
    then
        break
    fi
    echo "I like $fruit"
done
```

5. continue (조건식으로 복귀)

```
#!/bin/bash
count=1
while [ $count -le 5 ]
do
    if [ $count -eq 3 ]
    then
        count=$((count+1))
        continue
    fi
    echo "Count is $count"
    count=$((count+1))
done
```

쉘 스크립트 예제 – 함수, 위치 매개 변수

1. 함수

함수 정의(bash, dash 모두 실행 가능)

```
#!/bin/sh
say_hello () {
    echo "Hello, world!"
}
```

`say_hello`

함수 호출

3. 위치 매개 변수를 이용하는 함수

```
#!/bin/bash
function multiply {
    result=$(( $1 * $2 ))
    echo "The product of $1 and $2 is $result"
}
multiply 2 3
multiply 4 5
```

함수 호출 시에도 위치
매개 변수 활용 가능

2. 위치 매개 변수 (`$0, $1, $2...`)

```
#!/bin/bash
sum=$(expr $1 + $2)
echo "The sum of $1 and $2 is $sum"
```

`$1은 3, $2는 4`

산술 연산 수행

```
root@linux-test:~# ./sum.sh 3 4
The sum of 3 and 4 is 7
```

스크립트가 실행될 때 다음과 같이 위치
매개 변수를 통해 스크립트에 인자 전달

\$0	\$1	\$2
<code>./sum.sh</code>	3	4

쉘 스크립트 예제 – 특수 변수, 함수 지역 변수

1. 함수 지역변수

```
#!/bin/bash

my_function() {
    local my_var="Hello, world!" # 함수 내에서만 사용되는 지역 변수 선언
    echo $my_var
}

my_function # 함수 호출
```

함수 호출 시 생성, 함수 반환 시 소멸
쉘 변수, 환경 변수와 겹침 방지

2. 특수 변수

```
#!/bin/bash

echo "Script name: $0"
echo "PID of the script: $$"
echo "Number of arguments passed: $#"
echo "All arguments passed: $@"
echo "Exit status of the last command executed: $"
```

특수 변수 (Special Variables)

특수 변수	의미
\$\$	쉘 스크립트의 PID (Process ID)
\$#	전달된 인자의 개수
\$@	전달된 모든 인자
\$?	마지막 명령 실행 뒤 반환값

쉘 스크립트 예제 – 함수, 위치 매개 변수, 특수 변수

```
#!/bin/bash

function multiply {
    result=$(( $1 * $2 ))
    echo "The product of $1 and $2 is $result"
    return $result
}

multiply 2 3
result1=$?
echo "Return value is $result1"

multiply 4 5
result2=$?
echo "Return value is $result2"
```

앞선 내용을 바탕으로 스크립트의 동작을
유추해 보세요

환경 설정

사용자가 쉘을 사용하는 환경을 구성

사용자 로그인 시 실행되는 **쉘 스크립트**로 진행

환경 설정 파일

先
(먼저 적용)

시스템 환경
설정 파일

전체 사용자 공통 환경 설정

모든 사용자 로그인 시 무조건 실행

/etc/profile

/etc/bash.bashrc

/etc/profile.d/*.sh

가급적 수정하지 않는다

後
(이후 적용)

사용자 환경
설정 파일

사용자 별 환경 설정

각 사용자 홈 디렉토리에 숨김 파일로 존재

사용자가 수정 관리 가능

~/.profile

~/.bashrc

~/.bash_logout

~/.bash_aliases

환경 설정 변경 실습

```
PS C:\Users\magicnotebook> ssh root@106.10.32.76 -p 1024  
root@106.10.32.76's password:
```

로그인

Shell 실행

로그인 환경 설정 파일(**Shell 스크립트**) 실행

- 사용자 환경을 초기화 -

```
root@linux-test:~#
```

Shell 사용

}

쉘 사용 환경 적용

- 프롬프트 모양 변경 (외 여러가지) -

```
root@linux-test:~# exit
```

로그아웃

logout

Goodbye! Have a nice day!

Connection to 106.10.32.76 closed.

```
PS C:\Users\magicnotebook>
```

로그아웃 환경 설정 파일 실행

- 사용자 별 로그아웃 동작 수행 -

Shell 종료

환경 설정 변경 실습 - 사용자 프롬프트 모양 변경



리눅스 터미널 프롬프트에서 사용자 계정을 파란색, 호스트 부분을 빨간색으로 변경하고 싶어



리눅스 터미널 프롬프트에서 색상을 변경하려면, PS1 환경 변수를 수정해야 합니다. PS1은 프롬프트의 형식을 지정하는데 사용되는 환경 변수입니다.

1) ChatGPT에 질문

먼저, 다음 명령어를 사용하여 현재 PS1 값을 확인합니다.

bash

Copy code

```
echo $PS1
```

그린 다음 다음과 같이 PS1 값을 수정합니다.

less

Copy code

```
PS1='[\e[1;34m]\u\[e[m]@\[\e[1;31m\]\h\[e[m]:\w\$ '
```

2) 적용할 코드 복사

환경 설정 변경 실습 - 사용자 프롬프트 모양 변경

```
root@linux-test:~# nano .bashrc
```

3) .bashrc 파일을 텍스트 에디터로 오픈

```
GNU nano 2.9.3          .bashrc

alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# 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 [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi
```

PS1='[\e[1;34m]\u@\[\e[m\]@\[\e[1;31m\]\h[\e[m]:\w\\$ '

4) 파일 끝부분에 2)에서 복사한 내용 붙여넣기

5) 파일 저장

AG Get Help **W Write Out** **W Where Is** **C Cut Text** **J Justify** **C Cur Pos**
X Exit **R Read File** **Replace** **U Uncut Text** **T To Spell** **G Go To Line**

6) 종료

```
root@linux-test:~# source .bashrc
root@linux-test:~#
```

7) 수정된 환경 설정 파일 적용 (source 명령)

환경 설정 변경 실습 - 로그아웃 Goodbye 메시지 출력

```
root@linux-test:~# touch .bash_logout
```

1) .bash_logout 파일 생성

```
root@linux-test:~# nano .bash_logout
```

2) 에디터로 .bash_logout 파일 오픈

```
GNU nano 2.9.3          .bash_logout  
echo "Goodbye $USER. Have a nice day!"  
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

```
root@linux-test:~# exit  
logout  
Goodbye root. Have a nice day!  
Connection to 106.10.32.76 closed.  
PS C:\Users\magicnotebook>
```

4) exit 명령으로 로그아웃 시 메시지 출력 확인

Appendix 1 : 표준 입출력 - 리눅스

표준 입력 / 출력 / 에러에 대한 리눅스 장치 파일

```
root@linux-test:~# ls -al /dev/std*
lrwxrwxrwx 1 root root 15 Feb 27 09:17 /dev/stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 Feb 27 09:17 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 Feb 27 09:17 /dev/stdout -> /proc/self/fd/1
```

왜 심볼릭 링크로 되어 있나?

이것이 의미하는 것은?

생각해 보세요~*

Appendix 1 : 표준 입출력 - 리눅스

사용자 접속 환경에 따라 달라지는 리눅스 표준 입출력

```

bhahn@bhahn:~$ ls -al /dev/std*
lrwxrwxrwx 1 root root 15 Feb 27 02:18 /dev/stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 Feb 27 02:18 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 Feb 27 02:18 /dev/stdout -> /proc/self/fd/1
bhahn@bhahn:~$ ls -al /proc/self/fd/
total 0
dr-x----- 2 bhahn bhahn 0 Feb 27 02:32 .
dr-xr-xr-x 9 bhahn bhahn 0 Feb 27 02:32 ..
lrwx----- 1 bhahn bhahn 64 Feb 27 02:32 0 -> /dev/tty1
lrwx----- 1 bhahn bhahn 64 Feb 27 02:32 1 -> /dev/tty1
lrwx----- 1 bhahn bhahn 64 Feb 27 02:32 2 -> /dev/tty1
lr-x----- 1 bhahn bhahn 64 Feb 27 02:32 3 -> /proc/2425/fd
bhahn@bhahn:~$ _

```

물리적 콘솔로 연결했을 때

/dev/stdin -> /proc/self/fd/0 ->	/dev/tty1
/dev/stdout -> /proc/self/fd/1 ->	/dev/tty1
/dev/stderr -> /proc/self/fd/2 ->	/dev/tty1

네트워크(ssh)를 통해 연결했을 때

/dev/stdin -> /proc/self/fd/0 ->	/dev/pts/0
/dev/stdout -> /proc/self/fd/1 ->	/dev/pts/0
/dev/stderr -> /proc/self/fd/2 ->	/dev/pts/0

```

bhahn@bhahn:~$ ls -al /dev/std*
lrwxrwxrwx 1 root root 15 Feb 27 02:18 /dev/stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 Feb 27 02:18 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 Feb 27 02:18 /dev/stdout -> /proc/self/fd/1
bhahn@bhahn:~$ ls -al /proc/self/fd/
total 0
dr-x----- 2 bhahn bhahn 0 Feb 27 02:32 .
dr-xr-xr-x 9 bhahn bhahn 0 Feb 27 02:32 ..
lrwx----- 1 bhahn bhahn 64 Feb 27 02:32 0 -> /dev/pts/0
lrwx----- 1 bhahn bhahn 64 Feb 27 02:32 1 -> /dev/pts/0
lrwx----- 1 bhahn bhahn 64 Feb 27 02:32 2 -> /dev/pts/0
lr-x----- 1 bhahn bhahn 64 Feb 27 02:32 3 -> /proc/2430/fd

```

Appendix 1 : 표준 입출력 - 리눅스

tty와 pts의 차이 (by ChatGPT)



tty와 pts의 차이에 대해 알려줘



tty와 pts는 모두 Unix 계열 운영 체제에서 사용되는 가상 터미널(Virtual Terminal)을 나타내 는 용어입니다.

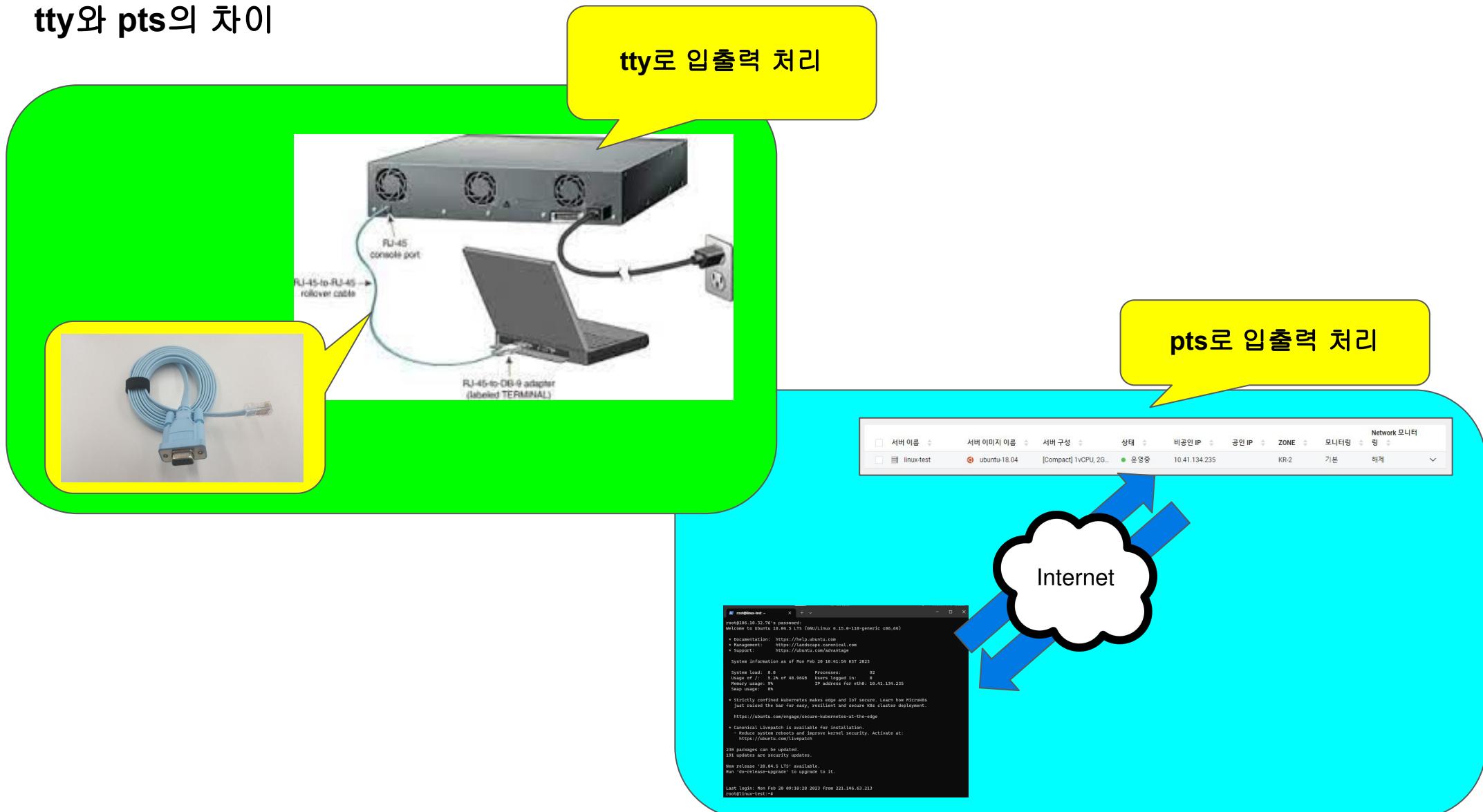
tty는 "teletype"의 약자로, 이전에는 터미널 장치와 직접적으로 연결되어 텍스트 기반의 사용자 인터페이스를 제공하는 터미널을 의미했습니다. 하지만 현재는 터미널 장치와 직접적인 연결이 없는 가상 터미널을 나타내는 용어로 사용됩니다. 일반적으로 tty는 콘솔 환경에서 사용되며, 사용자가 로그인하여 쉘을 실행할 수 있는 환경을 제공합니다.

반면, pts는 "pseudo terminal slave"의 약자로, 실제 하드웨어 장치를 사용하지 않고 가상의 터미널을 생성할 때 사용됩니다. pts는 일반적으로 ssh나 telnet과 같은 원격 접속 프로토콜을 사용하여 접속하는 경우에 사용됩니다. 이 경우, ssh나 telnet과 같은 클라이언트는 pts에 연결되며, 클라이언트에서 사용자 입력을 받아 처리하고, pts에 출력을 전달합니다.

즉, tty와 pts의 가장 큰 차이점은 tty가 콘솔 환경에서 사용되고, pts는 원격 접속 환경에서 사용된다는 것입니다. 또한, tty는 실제 하드웨어 터미널을 가리키는 경우가 많지만 pts는 가상 터미널을 가리키는 경우가 대부분입니다.

Appendix 1: 표준 입출력 - 리눅스

tty와 pts의 차이



Thank You