



ENTERPRISE LINUX ADMIN GUIDE

BASH SHELL

단원목표

- 리다이렉션(Redirection)
 - 파이프(Pipe)
 - 배쉬셸 기능(Bash Shell Funtion)
 - 변수(Variable)
 - 메타캐릭터(Metacharcter)
 - 히스토리(History)
 - 환경 파일(Environmmnet File)
-

셸은 명령행에서 입력되거나 스크립트 파일에서 읽어 들이는 명령어들을 해석함으로써 사용자와 커널의 중계 (Interface) 역할을 담당한다. 사용자가 로그인하면 셸이 시작되고 사용자가 명령어를 입력하면 셸은 명령행의 구문을 분석하고, 리다이렉션, 파이프, 와일드 카드, 작업 제어등을 처리하고, 명령어를 PATH변수에서 검색하고 존재하면 실행한다. 셸의 주요 기능들 중 하나는 명령행 프롬프트에서 입력된 명령어들을 해석하는 대화형 기능이다. 셸은 명령행에서 입력된 명령어들을 분석해서 '토큰'이라는 단어 단위로 해석한다. 토큰은 탭이나 빈칸, 개행문자 등 공백으로 구분한다. 입력된 명령어들에 메타문자들이 포함되어 있으면 적절하게 처리한다. 셸은 파일 입출력과 백그라운드 작업을 처리하며, 명령행에서 입력이 정상적으로 분석되었을 경우 해당 명령어를 찾아 실행한다.

셸의 기능을 요약해 보면 다음과 같다.

- 셸은 PATH 변수에 정의된 모든 디렉토리를 참고 하여 명령어를 찾아 실행한다.
- 셸은 파이프, 입/출력 리다이렉션, 백그라운드 프로세싱(Background Processing)을 설정한다.
- 셸은 명령어의 모임인 별칭(alias), 셸 함수(Function) 등을 찾아 실행시킨다.
- 셸은 TERM 변수를 사용하여 서로 다른 터미널 환경을 초기화 한다.
- 셸은 명령어 자동 완성 기능을 사용하여 입력한 명령어를 기록한다.
- 셸은 사용자 환경 정의 파일을 가지고 사용자의 환경을 초기화 한다.

다음과 같은 셸의 기능에 대해서 배운다.

■ 셸의 기능

- **명령어 해석기**(Command interpreter)
- **프로그램 언어**(Programable Language) ----> 셸 스크립트(셸 프로그램)

- 리다이렉션(Redirection)
- 파이프(Pipe)
- 셸 기능(Shell Function)
- 변수(Variable)
- 히스토리(History)
- 환경파일(Environment Files)

리다이렉션(Redirection)

- 파일기술자

- 입력 리다이렉션(Redirection stdin)

```
# mail -s "OK:linuxXXX" root < report.txt
```

- 출력 리다이렉션(Redirection stdout)

```
# ls -l > ls.txt
```

- 에러 리다이렉션(Redirection stderr)

```
# ./backup.sh > backup.log 2>&1
```

1 방향 재지정 (Redirection)

(1) fd (파일 기술자, File Descriptor)

각 프로세스를 위한 파일 기술자(File Descriptor)가 셸에 의해 만들어 진다. 한 개의 파일을 열면 그 연 파일을 나타내는 특별한 숫자를 셸이 만들어 내는데 이것이 파일 기술자이다. 이런 번호들 중 미리 예약 되어져서 파일을 열 때 주어지지 않는 번호가 있는데 아래 표와 같다.

파일 기술자(File Description)란?

- 프로세스가 파일을 열때 할당되는 번호
- 프로세스의 열린 파일을 구분할 때 사용하는 식별 번호

[참고] C 언어

```
# vi test.c
```

```
int fd;  
fd=open(.....);
```

■ 예약되어진 파일 기술자(File Descriptor)

파일기술자	축약의미	설 명
0	stdin	Standard Input, 표준입력 , 특별히 입력이 지정되지 않은 경우 키보드에서 입력을 받는다.
1	stdout	Standard Output, 표준출력 , 특별히 출력이 지정되지 않은 경우 정상적인 출력 결과를 모니터로 출력된다.
2	stderr	Standard Error, 표준에러 , 특별히 출력이 지정되지 않은 경우 에러의 출력 결과를 모니터로 출력된다.

파일 기술자에 대한 정보는 /proc 디렉토리 하위에 존재하는 각 프로세스의 PID(Process ID) 번호 디렉토리 하위에 fd 디렉토리에서 확인할 수 있고 또한 /dev/fd 디렉토리 하위에서도 파일 기술자에 번호를 확인할 수 있다. 하지만 번호들은 열어 볼 수는 없는 파일들이다. 다음은 파일기술자에 대한 확인을 하기 위한 테스트이다. 가상터미널1에서 /etc/passwd 파일을 열고 다른 가상터미널2에서 현재 동작중인 프로세스를 확인한 후 프로세스가 열고 있는 파일을 확인해 보면 새로운 파일 기술자(File Descriptor)가 생성된 것을 확인한 것이다.

```
[TERM1]
# vi /etc/passwd
```

```
[TERM2]
# ps -ef | grep vi          /* vi가 실행된 PID번호를 볼 수 있다 */
```

```
root      2077    2045  0 02:26 pts/10    00:00:00 vi /etc/passwd
root      2081    32326  0 02:26 pts/8      00:00:00 grep vi
```

```
# cd /proc/2077/fd          /* 단, 프로세스의 번호(2077)은 시스템 마다 번호가 다를 수 있다. */
# ls -l                    /* 파일 기술자 4번이 할당되어 열린 것을 확인 할 수 있다. */
```

```
합계 0
lrwx----- 1 root root 64 3월 17 02:27 0 -> /dev/pts/10
lrwx----- 1 root root 64 3월 17 02:27 1 -> /dev/pts/10
lrwx----- 1 root root 64 3월 17 02:26 2 -> /dev/pts/10
lrwx----- 1 root root 64 3월 17 02:27 4 -> /etc/.passwd.swp
```

```
# cd
```

```
[TERM1]
# vi /etc/passwd          /* vi 편집기 실행 닫기 */
```

```
:q
```

```
[TERM2]
# cd /proc/2077/fd        /* 실행중인 vi 편집기의 프로세스 번호 */
```

```
-bash: cd: /proc/2077/fd: 그런 파일이나 디렉토리가 없음
```

[EX] proc 디렉토리에 프로세스 번호 생성여부 확인

```
[TERM2]
# cd /proc
# ls > /test/proc_`date +%H%M`.txt
```

```
[TERM1]
# vi /etc/passwd
```

```
[TERM2]
# cd /proc
# ps -ef | grep vi
```

```
root      12926   8360  0 16:26 pts/4      00:00:00 vi /etc/passwd
root      12928   12761  0 16:26 pts/1      00:00:00 grep vi
```

```
# grep 12926 /test/proc_1415.txt
```

```
출력값 없음
```

```
# ls > /test/proc_`date +%H%M`.txt
# ls /test
```

```
proc_1415.txt    proc_1418.txt
```

```
# grep 12926 /test/proc_1418.txt
```

```
12926
```

```
[TERM1]
# vi /etc/passwd
```

```
:q
```

```
[TERM2]
# ls > /test/proc_`date +%H%M`.txt
# ls /test
```

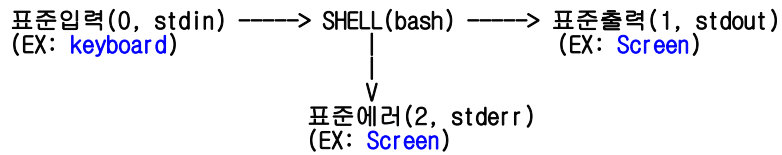
```
proc_1415.txt    proc_1418.txt    proc_1422.txt
```

```
# grep 12926 /test/proc_1422.txt
```

```
출력값 없음
```

(2) 표준 입력, 표준 출력

셸에서는 명령어 입력을 특별한 지정이 없는 한 키보드(Keyboard)에서 받는다. 그리고 명령어의 출력 결과를 모니터에 출력해준다. 이러한 명령어의 출력 결과나 입/출력을 리다이렉션 심볼(Redirection Symbol)을 사용하여 방향을 재지정 할 수 있다. 방향을 재 지정함으로 인해 기본 출력 방향이 모니터인 것을 file로 변경이 가능하다.



ls /var /nodir

```
ls: /nodir: 그런 파일이나 디렉토리가 없음
/var:
account  db      games  local  mail    opt      run      tux
cache    empty  gdm    lock   named   preserve spool    www
crash    ftp     lib     log    nis     racoon   tmp      yp
```

cat

```
hello<Enter>          /* 표준 입력(stdin) */
hello                 /* 표준 출력(stdout) */
Linux<Enter>
Linux
<Ctrl + D>
```

cat < /etc/passwd

cat > testfile

```
hello
Linux
<Ctrl + D>
```

cat testfile

```
hello
Linux
```

(3) 입력 재지정(입력 리다이렉션, Redirection stdin)

입력 재지정은 표준입력(Stdin)에서 입력을 받지 않고, 다른쪽에서 입력을 받는 것을 입력 재지정 (Redirection stdin)이라고 한다. 원래 특별한 지정이 없다면, 키보드에서 입력을 받지만 다른쪽 방향으로 입력을 받는 것을 말한다. 파일 기술자 0번은 생략 가능하다.

[형식]

```
CMD < filename          (표준입력)
CMD 0< filename         (표준입력 - 파일 기술자 0번)
```

[EX] 리다이렉션 실습

```
# cat < /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....
```

```
# cat 0< /etc/shadow
```

```
root:$1$GUT1Ey/2$ItYR1Bjp6er6klmf1/4DV1:14634:0:99999:7:::
bin*:14634:0:99999:7:::
daemon*:14634:0:99999:7:::
....
```

```
# mail user01          /* # mail user01@hanmail.net */
```

```
Subject: Test Mail
Hello Linux
CentOS 5.4 User
Cc: <ENTER>
```

```
# su - user01  (# mail -u user01)
$ mail
> #           (# is number)
> q
$ exit
```

```
# mail -s "Test Mail" user01 < /etc/hosts
# mail -s "linuxXXX : OK" admin@paran.com < report.txt
```

```
# wall          (# wall "System Migration Works with /dev/sda1")
"System Migration Works with /dev/sda1"
<CTRL + D>
# wall < /etc/hosts
```

(4) 출력 재지정(출력 리다이렉션, Redirection stdout)

출력 재지정은 정상적인 출력 결과를 표준출력(모니터)으로 출력을 하지 않고, 다른 쪽으로 출력하는 것을 출력 재지정(Redirection stdout)이라고 한다. 원래 특별한 지정이 없다면, 모니터로 출력되지만 다른 쪽 방향으로 출력하는 것을 말한다. 파일 기술자 1번은 생략 가능하다.

[명령어 형식]

```
CMD > filename          /* 표준출력, 덮어쓰기 기능 */
CMD 1> filename         /* 표준출력, 덮어쓰기 기능 */
CMD >> filename         /* 표준출력, 이어쓰기 기능 */
CMD 1>> filename        /* 표준출력, 이어쓰기 기능 */
```

[EX1] 리다이렉션(Redirection stdout) 실습

```
# cd
# ls -l
```

```
drwxr-xr-x 2 root root 4096 3월 15 15:55 Desktop
-rw----- 1 root root 4210 1월 25 22:54 anaconda-ks.cfg
-rw-r--r-- 1 root root 54631 1월 25 22:54 install.log
-rw-r--r-- 1 root root 9641 1월 25 22:50 install.log.syslog
```

```
# ls -l > lsfile      (# ls -l 1> lsfile)
```

```
# cat lsfile
```

```
drwxr-xr-x 2 root root 4096 3월 15 15:55 Desktop
-rw----- 1 root root 4210 1월 25 22:54 anaconda-ks.cfg
-rw-r--r-- 1 root root 54631 1월 25 22:54 install.log
-rw-r--r-- 1 root root 9641 1월 25 22:50 install.log.syslog
-rw-r--r-- 1 root root 0 3월 17 02:32 lsfile
```

[EX2] > , >> 차이점

```
# echo 1111
```

```
1111
```

```
# echo 1111 > filename1
```

```
# cat filename1
```

```
1111
```

```
# echo 2222 > filename1
```

/* 덮어쓰기 기능 */

```
# cat filename1
```

```
2222
```

```
# echo 1111 >> filename1
```

/* 이어쓰기 기능 */

```
# cat filename1
```

```
2222
```

```
1111
```

(5) 에러 재지정(에러 리다이렉션, Redirection stderr)

에러 재지정은 표준 에러로 출력하지 않고, 다른 쪽으로 출력하는 것을 에러 재지정(Redirection stderr)이라고 한다. 원래 특별한 지정이 없다면, 모니터로 출력을 하지만 다른 쪽으로 출력하는 것을 말한다. 파일 기술자 2번은 생략 불가능하다. 방향 재지정자를 1번과 동일하게 사용하여 1번 파일 기술자와 구별하기 위해 생략 불가능하다.

형식]

```
CMD 2> filename /* 표준에러출력 */
CMD 2>> filename /* 표준에러출력 */
```

[EX] 표준 에러 실습

```
# cd /test ; rm -rf /test/*
# touch file1 file2 file3
```

```
# ls -l /test /nodir
```

```
-----
정상출력
에러출력
-----
```

```
# ls -l /test /nodir > dirfilename1
```

```
-----
정상출력
에러출력
-----
```

```
# cat dirfilename1
```

-rw-r--r--	1	root	root	0	3월	17	02:34	dirfilename1
-rw-r--r--	1	root	root	0	3월	17	02:33	file1
-rw-r--r--	1	root	root	0	3월	17	02:33	file2
-rw-r--r--	1	root	root	0	3월	17	02:33	file3

```
# ls -l /test /nodir 2> dirfilename2
```

```
-----
정상출력
에러출력
-----
```

```
# cat dirfilename2
```

ls: /nodir: 그런 파일이나 디렉토리가 없음

```
# ls /test /nodir > dirfilename1 2> dirfilename2
```

```
-----
정상출력
에러출력
-----
```

```
# cat dirfilename1
```

```
# cat dirfilename2
```

```
# ls /test /nodir > dirfilename 2>&1
# cat dirfilename
```

(실무 예) script.sh(스크립트) 로그 파일 생성

(X) # ./script.sh > file.log

(O) # ./script.sh > file.log 2>&1

(실무 예) ./configure --prefix=/usr/local/apache2

./configure --prefix=/usr/local/apache2 2> file.log

./configure --prefix=/usr/local/apache2 > file.log 2>&1

(실무 예) 일반 사용자(EX: oracle, wasuser)가 검색하는 경우

\$ find / -name core -type f

\$ find / -name core -type f 2>/dev/null

파이프(Pipe)

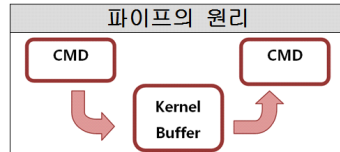
- CMD | CMD

```
# ps -ef | more
# ps -ef | grep xinetd
# CMD > file.log (# CMD >> file.log )
# CMD | tee file.log      (# CMD | tee -a file.log)
```

2 파이프 (Pipe)

파이프(Pipe)란? 앞에 실행한 명령어의 출력 결과를 뒤에 실행하는 명령어의 입력 값으로 넣어 준다. 앞의 출력 결과를 뒤에 실행하는 명령어의 입력 결과로 보내줌으로 인해 파이프(|)를 사용하면 여러 명령을 동시에 연결하여 사용할 수가 있다.

ls -al 명령을 사용하게 되면 모든 파일을 보여주므로 한 화면을 넘는 많은 파일을 보여 주게 될 때 "ls -al | more" 를 쓰게 되면 ls 명령에 대한 결과를 more 명령을 통해 확인해 볼 수 있다.

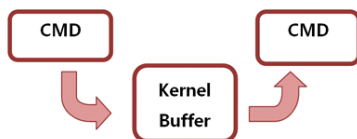


첫 번째 CMD로 출력된 내용을 화면에 그대로 출력하지 않고 커널 버퍼에 데이터를 저장해 두었다가 버퍼에 저장된 결과 값을 가지고 뒤에 있는 CMD의 입력 값으로 넘겨 뒤에 있는 명령어가 실행한 결과를 출력해 주게 된다.

[명령어 형식]
CMD | CMD

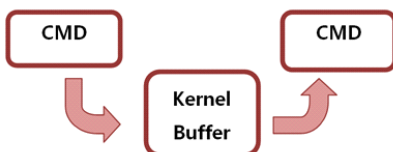
[EX] '|' (Pipe) 실습
ps -ef
ps -ef | less

/* 프로세스의 모든 정보 출력 */
/* CMD | less : 한 페이지가 넘는 페이지 출력 방법 지정 */

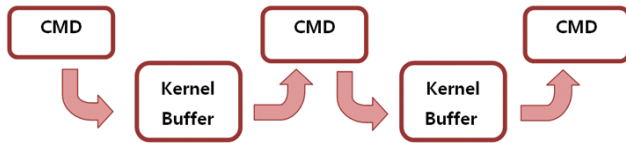


ps -ef | grep inetd

/* CMD | grep inetd 앞단의 명령어의 출력 결과 중 패턴 검색 */



ls -al / | sort -r | more /* CMD | sort를 하여 buffer에 넣고 buffer 내용을 한 화면 단위로 출력 */



ls 명령을 통해 출력된 결과를 sort 명령으로 정렬하고 more명령을 통해 보기 쉽게 출력하는 세 가지 명령을 사용하고 있다.

[참고] 명령어 비교

# ps -ef > file.log # grep inetd file.log	# ps -ef grep inetd
--	-----------------------

[참고] Pipe와 함께 사용 할 수 있는 명령어

- tee란? 파이프 중간에 사용하여 입력을 출력으로 보내기 전에 파일로 기록

man tee

개요

tee [-ai] [--append] [--ignore-interrupts] [--help] [--version] [파일...]

설명

이 맨페이지는 GNU 버전의 tee 를 다룬다. tee 명령은 표준입력으로부터의 입력을 표준출력 또는 인수로 주어진 파일에 복사한다.

씩 여질 파일이 존재하지 않으면 생성된다. 만약 이미 존재한다면 -a 옵션을 사용하지 않는 한 이전의 자료는 모두 지워진다.

옵션

-a, --append
덮어쓰지 않고 주어진 파일에 표준입력을 추가한다.

[EX] 출력 결과를 파일로 저장

```
# CMD > file.log
# CMD | tee file.log
```

```
# cal 2002 > file.log
# cal 2002 | tee file2.log
```

[EX1] 실시간 모니터링 하면서 파일로 내용을 저장

```
# while true
> do
> ps -ef | grep httpd | wc -l | tee -a file.log      (> w root >> file.log)
> sleep 2
> done
```

```
17:05:51 up 1 day,  7:12,  2 users,  load average: 0.00, 0.01, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
root      pts/2    172.16.7.1      16:00    0.00s  0.28s  0.02s w root
root      :0       -               12May10  ?xdm?  2:39   0.42s /usr/bin/gnome-
17:05:53 up 1 day,  7:12,  2 users,  load average: 0.00, 0.01, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
root      pts/2    172.16.7.1      16:00    2.00s  0.26s  0.00s w root
root      :0       -               12May10  ?xdm?  2:39   0.42s /usr/bin/gnome-
```

cat file.log

```
17:05:51 up 1 day,  7:12,  2 users,  load average: 0.00, 0.01, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
root      pts/2    172.16.7.1      16:00    0.00s  0.28s  0.02s w root
root      :0       -               12May10  ?xdm?  2:39   0.42s /usr/bin/gnome-
17:05:53 up 1 day,  7:12,  2 users,  load average: 0.00, 0.01, 0.00
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
root      pts/2    172.16.7.1      16:00    2.00s  0.26s  0.00s w root
root      :0       -               12May10  ?xdm?  2:39   0.42s /usr/bin/gnome-
```


[EX2] 다른 터미널 창에서 입력하는 내용이 그대로 지정한 터미널 창으로 출력 되도록 설정

```
[참고] script CMD
# cd /test ; rm -rf /test/*
# script -a file.log
# ls
# date
# cal
# exit
# cat file.log
```

[TERM1] 첫번째 윈도우

```
# tty
```

```
/dev/pts/1
```

→ TERM3 에서 입력하는 내용과 출력하는 내용이 그대로 화면에 출력 된다.

[TERM2] 두번째 윈도우

```
# tty
```

```
/dev/pts/2
```

[TERM3] 세번째 윈도우

```
# script -a /dev/null | tee /dev/pts/1 | tee /dev/pts/2
# ls
# date
# cal
# exit
```

배쉬셸 기능(Bash Shell Funtion)

- bash shell function

```
# set -o
# set -o vi
# set +o vi

# set -o ignoreeof
# set -o noclobber
```

3 bash 셸의 기능(bash Shell Function)

```
# set -o      /* 셸 자체의 기능 전체 목록 확인 */
# set -o vi   /* 셸 자체의 기능 중 vi 기능을 ON */
# set +o vi   /* 셸 자체의 기능 중 vi 기능을 OFF */
```

bash 셸의 기능

all export	off	
brace expand	on	
emacs	on	/* 기본적인 편집기 선언 */
errexit	off	
errtrace	off	
functrace	off	
hashall	on	
histexpand	on	
history	on	
ignoreeof	off	/* 로그아웃 방지 기능 선언 */
interactive-comments	on	
keyword	off	
monitor	on	
noclobber	off	/* 덮어쓰기 방지 기능 선언 */
noexec	off	
noglob	off	
nolog	off	
notify	off	
nounset	off	
onecmd	off	
physical	off	
pipefail	off	
posix	off	
privileged	off	
verbose	off	
vi	off	/* 기본적인 편집기 선언 */
xtrace	off	

(1) 로그아웃 <Ctrl+D> 방지하기 (ignoreeof)

셸 프롬프트 상태에서 <Ctrl+D>의 역할은 파일의 끝을 알려주거나 현재 셸프롬프트를 종료하는 기능을 가지고 있다. 잘못하여 <Ctrl+D>를 누르게 되면 관리자가 작업하던 중 셸프롬프트가 종료되면서 로그아웃이 되어 버린다. 관리자가 작업 중에 <Ctrl+D>를 눌러 로그아웃이 되는 것을 막기 위해 셸의 기능중 ignoreeof를 이용하여 로그아웃 방지 기능을 설정 하도록 한다. <Ctrl+D>를 이용한 로그아웃 방지 기능이 설정되면 exit를 이용하여 로그아웃을 하도록 한다.

```
set -o ignoreeof
```

EOF(End Of File)의 의미를 갖는 키 <CTRL + D>

- ① 파일의 끝(EOF)
- ② 현재 셸(프로그램) 종료

[EX1] 로그아웃 방지 기능

```
# telnet 172.16.9.252
```

```
Trying 172.16.9.252...
Connected to 172.16.9.252 (172.16.9.252).
Escape character is '^['.

linux252.example.com (Linux release 2.6.18-164.el5 #1 SMP Thu Sep 3 03:33:56 EDT 2009) (4)

login: root
Password: (root 사용자 암호 입력)
Last login: Mon Sep 19 10:38:26 from linux213
```

```
# hostname
# id
# ps
# <CTRL + D>
```

```
# telnet 172.16.9.252
root 사용자로 로그인
```

```
# set -o | grep ignoreeof
```

```
ignoreeof      off
```

```
# set -o ignoreeof      /* ignoreeof 기능 on */
# <CTRL + D>             /* 로그아웃 기능 */
```

```
use "logout" to leave the shell
```

```
# exit
#
```

【참고】 사용자 계정의 자동 로그아웃 기능

윈도우에서 사용자가 아무런 동작을 하지 않으면 자동 로그아웃이 되는 기능이 있듯이 Linux에서도 사용자가 지정된 시간동안 아무런 동작을 하지 않으면 자동 로그아웃이 되는 기능이 있다.

▶ /etc/profile 환경 파일 : 모든 사용자에게 적용가능

```
# vi /etc/profile
```

```
...
HOSTNAME='/bin/hostname'
...
TMOUT=3600          /* 자동 로그아웃 되는 시간 : 초단위로 지정 */
...
```

▶ .bash_profile 환경파일 : 사용자별 홈 디렉토리에 있는 파일을 변경

```
# vi ~/.bash_profile
```

```
...
TMOUT=60           <-- 라인 추가
export PATH TMOUT  /* 지역변수를 환경 변수로 적용 */
...
```

(2) 덮어쓰기 방지기능 (noclobber)

방향 재 지정자(Redirection Stdout, '>')를 이용하면 기존에 존재하는 파일이름에 새로운 내용으로 덮어쓰기가 가능했었다. 중요한 파일의 경우 파일의 이름이 존재하는지 모른 채 덮어쓰기를 할 우려가 있기 때문에 덮어쓰기 방지 기능을 이용하여 중요한 파일등에 덮어쓰기가 되지 않도록 파일의 내용을 보호 할 수가 있다.

```
set -o noclobber
```

```
# set -o | grep noclobber
```

```
noclobber      off
```

```
# set -o noclobber          /* 덮어쓰기 방지 기능 on */  
# set -o | grep noclobber
```

```
noclobber      on
```

```
# echo 1111 > test1  
# echo 2222 > test1          /* 덮어쓰기가 되지 않는다 */
```

```
-bash: test1: cannot overwrite existing file
```

```
# cat test1
```

```
1111
```




```
# set +o noclobber          /* 덮어쓰기 방지 기능 off */  
# echo 2222 > test1  
# cat test1
```


```
2222
```

```
# set -o noclobber          /* 덮어쓰기 방지 기능 on */  
# echo 3333 >| test1        /* 덮어쓰기 방지 무시 */
```

```
3333
```

4 <Tab> 파일 이름 자동 완성 기능

디렉토리에 있는 파일이나 디렉토리의 첫 번째 문자만 입력 후 Tab()키를 누르면 첫 글자로 시작하는 파일이나 디렉토리의 나머지 글자가 자동으로 완성된다. 단, 첫 문자가 같은 디렉토리나 파일이 있는 경우 Tab()키 한번만 누르면 아무 변화도 없지만 Tab()키 두 번 연속으로 누르면 동일하게 시작하는 모든 디렉토리나 파일을 출력해 준다.

[명령어 형식]
ls t 

[EX1] <TAB> 이용한 디렉토리 이동

```
# cd /etc/sysco<TAB>
```

```
/etc/sysconfig/
```

```
# cd /etc/sysconfig/netw<TAB>
```

```
/etc/sysconfig/network
```

```
# cd /etc/sysconfig/network-s<TAB>
```

```
/etc/sysconfig/network-scripts/
```

[EX2] 자동 완성 기능이 되지 않을때의 <TAB><TAB> 사용

```
# cd /etc/sys<TAB><TAB>
```

```
sysconfig/ sysctl.conf syslog.conf
```

```
# cd /etc/sysco<TAB> /* 다른 문자와 구별 할 수 있는 글자까지 적어주고 다시<Tab>을 누르면 완성 */
```

```
/etc/sysconfig
```

[EX3] <TAB> 이용한 명령어의 종류 확인

```
# ch<TAB><TAB>
```

chac1	check-binary-files	chmod
chage	checkXML	chown
change_console	checkmodule	chpasswd
charmap	checkpolicy	chroot
chat	chfn	chrt
chattr	chgrp	chsh
chcat	chkconfig	chvt
chcon	chkfontpath	

```
# system-config-<TAB><TAB>
```

system-config-authentication	system-config-nfs
system-config-date	system-config-packages
system-config-display	system-config-printer
system-config-httpd	system-config-rootpassword
system-config-kdump	system-config-samba
system-config-keyboard	system-config-securitylevel
system-config-language	system-config-securitylevel-tui
system-config-lvm	system-config-services
system-config-network	system-config-soundcard
system-config-network-cmd	system-config-time
system-config-network-gui	system-config-users
system-config-network-tui	

[참고] redhat-config-* & system-config-*

REEL 3.X(CentOS 3.X) : redhat-config-
REEL 4.X(CentOS 4.X) : system-config-
REEL 5.X(CentOS 5.X) : system-config-
REEL 6.X(CentOS 6.X) : system-config- (기본설치X)
REEL 7.X(CentOS 4.X) : ??


```
# set -o vi
(↵) 파일을 자동 완성(File name completion)
(⌘) 명령행 편집(Command Line Edit)
```

(↵) 파일 이름 자동 완성 기능

```
# cd /etc/sysco[TAB]
# cd /etc/sysco[ESC][w]
```

(⌘) 명령행 편집
(이전 명령어를 되살려서 쓰는 경우)

```
# ↑↑↑↓↓↓
# [ESC][k][k][k][j][j][j]
```

(이전 명령어를 편집해서 사용하는 경우)

```
# find / -name core -type f
<CTRL + C>
# [ESC][k]
# find / -name file1 -type f
```

```
# cp /etc/service file1
# cp file1 file2
# cp file1 file3
# cp file1 file4
# cp file1 file5
```

(이전에 사용한 명령어를 확장해서 사용하는 경우)

```
# df -h
# df -h /
# df -h / | tail -1
# df -h / | tail -1 | awk '{print $5}'
```

```
# ps -ef | head
# ps -ef | head | sort -k 2
# ps -ef | head | sort -k 2 -n
# ps -ef | head | sort -k 2 -nr
```

(확인 + 명령어 수행 + 확인)

```
# chkconfig --list krb5-telnet
# chkconfig krb5-telnet on
# ↑↑
```

변수(Variable)

- 변수의 종류
지역 변수(Local Variable)
환경 변수(Environment Variable)
특수 변수(Special Variable)
- 변수 선언 방법
VAR=5 ; export VAR
echo \$VAR
unset VAR
- 변수 export란?

5 변수

변수의 종류

- 지역변수(Local Variable) : # VAR=5
- 환경변수(Environment Variable) : # export VAR=5
- 특수변수(Special Variable) : \$\$, \$?, \$!, \$0, \$1, \$#, \$*,

(1) 지역변수 선언

변수 선언 방법(EX: bash)

```
# VAR=5      (# export VAR=5)
# export VAR
# echo $VAR   (# print $VAR)
# unset VAR
```

지역 변수는 현재 사용하고 있는 셸에만 적용되는 변수 값으로 선언 하는 것이다. 서브 셸에서 지역 변수 값을 확인하려고 할 경우 값이 나타나지 않는다.

```
var=5      echo $var (0)
-----+-----
|                   |
| echo $var (X)    |
|                   |
+-----+
```

```
# var=5
# echo $var
```

5

```
# var=hello
# echo $var
```

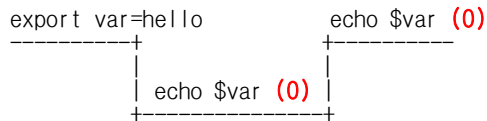
hello

```
# bash
# echo $var
#
```

```
# exit
# echo $var
```

(2) 환경변수 선언

환경변수는 **현재 셸과 서브셸에 변수를 모두 적용되는 변수 값으로 선언해 주는 것**이다. 환경변수를 사용하는 이유는 이전셸에서 선언한 변수 값을 서브셸에서도 동일하게 적용하기 위한 것이다.



```
# var=hello
# export var
# echo $var
```

```
hello
```

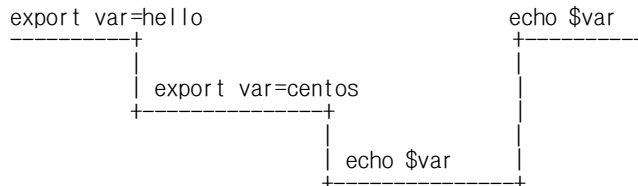
```
# bash
# echo $var
```

```
hello
```

```
# exit
```

```
# echo $var
```

```
hello
```



```
# var=hello      (# export var=hello)
# export var
# echo $var
```

```
hello
```

```
# bash
# echo $var
```

```
hello
```

```
# export var=centos
# echo $var
```

```
centos
```

```
# bash
# echo $var
```

```
centos
```

```
# ps
```

PID	TTY	TIME	CMD
14072	pts/2	00:00:00	bash
14075	pts/2	00:00:00	bash
14078	pts/2	00:00:00	bash

```
# exit
# exit
# echo $var
```

```
hello
```

(결론) 환경 변수는 자신이 선언한 셸의 서브셸에서 적용되는 것이다. 상위 셸에는 적용되지 않는다.

[참고] env와 set의 차이점 (선언된 변수를 확인하는 명령어)

- set : 모든 변수에 대해 출력 (지역변수 + 환경변수)
- env : 환경변수만 출력

[EX] set/env 명령어 실습

```
# var1=CentOS
```

```
# export var2=Linux
```

```
# set | grep var1
```

```
var1=CentOS
```

```
# set | grep var2
```

```
var2=Linux
```

```
# bash
```

```
# env | grep var1
```

```
#
```

-> 출력되는 정보가 없다.

```
# env | grep var2
```

```
var2=Linux
```

(3-1) 시스템 지역변수

i) PS1 명령어 프롬프트에 나타내는 시스템 환경 변수

```
#
```

```
# echo $PS1
```

```
[Wu@Wh Ww]W$
```

```
[root@linux203 ~]#  
PS1='[Wu@Wh Ww]W$ '
```

```
Wu   : username  
Wh   : hostname  
Ww   : working directory (예: /home/user01 -> user01)  
Ww   : working directory (예: /home/user01 -> /home/user01)  
W$   : root(#), user01($)
```

■ 현재 경로만 표시

```
# PS1='Ww> ' /* 윈도우 처럼 절대 경로가 출력 '[절대경로]> ' */
```

```
/root>
```

```
# PS1='$PWD> '  
# PS1='Ww> '
```

```
# set | grep PS1
```

```
$PWD>
```

```
# env | grep PS1
```

```
#
```

```
/root> bash
```

```
[root@linux200 ~]#
```

```
# PS1='[$PWD]# '
```

```
[/root]#
```

```
# PS1='[$PWD]# '  
# PS1='[Ww]# '
```

```
[/root]# bash
```

```
[root@linux200 ~]#
```

```
# export PS1='[Wu@$PWD]# '
```

```
[root@/root]#
```

```
# export PS1='[Wu@$PWD]# '  
# export PS1='[Wu@Ww]W$ '
```

```
# bash
```

```
[root@/root]#
```

```
# export PS1='[Wu@Wh Ww]W$ '
```

```
[root@linux200 ~]#
```

```
# exit
```

```
# exit
```

```
# exit
```

```
#
```

(실무 예) PS1='[Wu@Wh Ww]W\$ '

```
# vi ~/.bashrc
```

```
.....
```

```
#  
# Variable Definition  
#
```

```
export PS1='[Wu@Wh Ww]W$ '
```

```
# . ~/.bashrc (# source ~/.bashrc)
```

[참고] PS1 시스템 환경 변수 설정하는 방법

셸 프롬프트에 보여지는 항목을 설정하는 것으로 아래와 같은 것들을 사용할 수 있다.
 ex) [로그인한 계정@호스트네임 현재경로] # /* 현재 상태 \$: 프롬프트(\$, root일 경우 #) */

Wt 현재 시간을 HH:MM:SS 형식으로 표시
 Wd 날짜를 "요일 월 일" 형식으로 표시 (예, "Tue May 26")
 Wn 개행문자
 Ws 셸의 이름, \$0 의 베이스 이름 (마지막 슬래쉬 뒤 부분)
 Ww 현재 작업 디렉토리
 WW 현재 작업 디렉토리의 베이스 이름
 Wu 현재 사용자의 사용자명
 Wh 호스트 이름
 W# 이 명령의 명령 번호
 W! 이 명령의 히스토리 번호
 W\$ 유효 UID가 0 이면 if the effective UID is 0, a #, 그렇지 않으면 \$
 Wnnn 팔진수 nnn에 해당하는 문자
 WW 백슬래쉬
 W[비출력 문자의 시퀀스를 시작한다. 프롬프트에 터미널 제어시퀀스를 넣을때 사용한다.
 W] 비출력 문자의 시퀀스를 마친다.

ii) PS2 명령어가 아직 끝나지 않았음을 나타낼 때 사용

```
PS2='> '
# echo $PS2
>
# ls -a -l
# ls -a W
> -l

# echo "hello linux"
> "

# while true
> do
> CMD
> sleep 5
> done
```

echo \$PS2

>

(3-2) 시스템 환경변수

echo \$HOME

/root

cd \$HOME

pwd

/root

echo \$PWD

/root

echo \$LOGNAME

root

echo \$TERM

xterm

echo \$LANG

ko_KR.UTF-8

echo \$USER

root

echo \$UID

0

(4) 알아두면 좋은 특수 변수

① \$: 현재셸의 PID를 저장하고 있다. 쉘 스크립트내에서 임시 파일의 이름을 지정할 때 보통 사용이 된다.

echo \$\$

```
11991
```

ps

```
PID TTY      TIME CMD
11991 pts/1    00:00:00 bash
```

② ? : 바로 이전 명령어의 정상 실행 여부에 대한 결과값이 들어 있다. 쉘 스크립트내에서 이전 명령어의 정상 수행 여부를 확인할 때 주로 사용된다.

ls /nodir

```
ls: /nodir: No such file or directory
```

echo \$?

```
2
```

cd /nodir

```
-bash: cd: /nodir: No such file or directory
```

echo \$?

```
1
```

no

```
-bash: no: command not found
```

echo \$?

```
127
```

ls

```
Desktop      hash.tar.gz      mbox           top.seceret.mail.gpg
anaconda-ks.cfg  install.log      serv.pub
hash.md5      install.log.syslog  top.seceret.mail
```

echo \$?

```
0
```

(실무 예) 백업 스크립트 작성시

vi backup.sh

```
tar cvzf /backup/backup.tar.gz /home
if [ $? -eq 0 ] ; then          /* -eq : equal */
    echo "success"
else
    echo "error"
fi
```


[참고] /root/bin/script.sh 편하게 실행하기

- 관리자가 생성한 셸 스크립트(셸 프로그램)들을 모아 놓은 디렉토리
 - /root/bin or /root/shell

echo \$PATH

```
/usr/lib/qt-3.3/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
```

PATH : 명령어들이 위치한 디렉토리를 선언할 때 사용하는 변수

```
# ls -al
(절대경로) # /bin/ls -al
(상대경로) # cd /bin
# ./ls -al
```

```
# mkdir -p /backup
# mkdir -p /root/bin
# vi /root/bin/backup.sh
```

```
#!/bin/bash

tar cvzf /backup/back-`date +%m%d`.tar.gz --absolute-name /test
```

chmod 755 /root/bin/backup.sh

```
# export PATH=$PATH:/root/bin
# backup.sh
# cd /backup ; ls
```

```
back-0520.tar.gz
```

[참고] 백업 스크립트 실패 유무 확인

```
# cd /test
# vi script.sh
```

```
#!/bin/bash

/root/bin/backup.sh
if [ $? -eq 0 ] ; then          /* -eq : equal */
    echo "Backup Success"
else
    echo "Backup Fail"
fi
```

```
# chmod 755 /test/script.sh
# ./script.sh
```

```
/test/
/test/script.sh
Backup Success
```

vi /root/bin/backup.sh

```
#!/bin/bash

tar cvfz /backup/back-`date +%m%d`.tar.gz --absolute-name /nodir
```

./script.sh

```
tar: /nodir: Cannot stat: 그런 파일이나 디렉토리가 없음
tar: Error exit delayed from previous errors
Backup Fail
```

```
# ./script.sh > backup.log 2>&1
# cat backup.log
```

```
tar: /nodir: Cannot stat: 그런 파일이나 디렉토리가 없음
tar: Error exit delayed from previous errors
Backup Fail
```

③ ! : 바로 이전에 백그라운드로 실행한 프로세스의 PID번호가 저장 된다.

sleep 300 &

[1] 25192

[1] : Job ID
25192 : PID

echo \$!

25192

ps

PID	TTY	TIME	CMD
24868	pts/1	00:00:00	bash
25101	pts/1	00:00:00	bash
25192	pts/1	00:00:00	sleep
25219	pts/1	00:00:00	ps

sleep 400 &

[2] 25233

ps

PID	TTY	TIME	CMD
24868	pts/1	00:00:00	bash
25101	pts/1	00:00:00	bash
25192	pts/1	00:00:00	sleep
25233	pts/1	00:00:00	sleep
25234	pts/1	00:00:00	ps

echo \$!

25233

④ 인자변수(Argument Variable)

\$로 표시하여 인자변수의 값이 몇 번째인지 나타낼 수 있다.

인자변수1

인자변수2

인자변수3

```
# ls -a
# vi ls
```

```
..... (중략) .....
echo "$1"
..... (중략) .....
```

```
# cd /test
# vi test.sh
```

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

```
echo $1          /* 첫번째 인자 */
echo $2          /* 두번째 인자 */
echo $3          /* 세번째 인자 */
```

```
# chmod 755 test.sh
# ls -l test.sh
```

```
-rwxr-xr-x 1 root root 38 Aug 19 14:22 test.sh
```

```
# ./test.sh test1 test2 test3
```

```
test1
test2
test3
```

```
# date
```

```
Thu Aug 19 14:24:33 KST 2010
```

```
# set $(date)
# echo $1
```

```
Thu
```

```
# echo $2
```

```
Aug
```

```
# echo $3
```

```
19
```

```
# echo $4
```

```
14:24:47
```

```
# echo $5
```

```
KST
```

```
# echo $6
```

```
2010
```

(5) PATH 변수

명령어를 검색할 디렉토리를 선언할 때 사용하여 절대 경로를 쓰지 않고 실행파일명만을 이용해서 사용할 수 있게 한다. which라는 명령도 PATH에 설정되어 있는 디렉토리의 파일의 경로만을 알려준다.

```
# gedit &
# which gedit
```

```
/usr/bin/gedit
```

```
# echo $PATH
```

```
/usr/lib/qt-3.3/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/root/bin
```

```
# cd
# ls -l /test/script.sh
```

```
-rwxrwxrwx 1 root root 109 5월 20 15:26 /test/script.sh
```

(필요하면 아래와 같이 /test/script.sh 스크립트를 생성한다.)

```
# vi /test/script.sh
```

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

```
echo "script.sh Excution"
```

```
# chmod 755 /test/script.sh
```

```
# script.sh
```

```
-bash: script.sh: command not found
```

```
(절대경로)      # /test/script.sh
(상대경로)      # cd /test
                # ./script.sh
```

```
# script.sh
```

```
# vi ~/.bash_profile
```

```
# .bash_profile
```

```
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
```

```
# User specific environment and startup programs
```

```
PATH=$PATH:$HOME/bin
```

```
PATH=$PATH:/test <----- 라인 추가
```

```
export PATH
unset USERNAME
```

```
# . ~/.bash_profile
# echo $PATH
```

```
/usr/lib/qt-3.3/bin:/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin:/root/bin:/test
```

```
# script.sh
```

```
script.sh Excution
```

(6) HISTTIMEFORMAT 변수

```
# export LANG=C
# man bash
```

```
/HISTTIMEFORMAT
HISTTIMEFORMAT
    If this variable is set and not null, its value
    is used as a format string for strftime(3) to
    print the time stamp associated with each history
    entry displayed by the history builtin. If this
    variable is set, time stamps are written to the
    history file so they may be preserved across
    shell sessions.
```

```
# man 3 strftime
```

```
/%F      %F      Equivalent to %Y-%m-%d (the ISO 8601 date for-
mat). (C99)
/%T      %T      The time in 24-hour notation (%H:%M:%S). (SU)
```

```
# vi /etc/profile
```

```
..... (중략) .....
for i in /etc/profile.d/*.sh ; do
  if [ -r "$i" ]; then
    if [ "${-#*i}" != "$-" ]; then
      . $i
    else
      . $i >/dev/null 2>&1
    fi
  fi
done

#
# (1) Sfecific Configuration
#
export HISTTIMEFORMAT="%F %T "

unset i
unset pathmunge
```

```
# telnet localhost
root 사용자로 로그인
```

```
# history
```

```
..... (중략) .....
776 10:07:00 clear
777 10:07:11 clear
778 10:10:42 clear
779 10:12:43 export LANG=C
780 10:13:12 man bash
781 10:15:13 clear
782 10:15:16 man 3 strftime
783 10:17:49 vi /etc/profile
784 10:20:15 history
```

```
# exit
#
```

■ 메타캐릭터(Metacharacter)

`'', '"', '\', \w, ;`

- `'` (작은 따옴표(single quotation))
- `"` (큰 따옴표(Double quotation))
- ``` (역 따옴표(Back quotation))
- `\w` (역 슬래쉬(Back slash))
- `;` (세미콜론(Semicolon))

■ `'` (작은 따옴표(single quotation))

셸이 해석 할 수 없도록 막아 준다.

```
# echo $HOME
# echo '$HOME'
```

■ `"` (큰 따옴표(Double quotation))

셸이 해석 할 수 없도록 막아 준다. 단 인식되는 문자(`$`, ```, `\w`)들도 있다.

```
# echo $HOME
# echo "$HOME"
# echo "$HOME is my    directory.???"
```

■ ``` (역 따옴표(Back quotation))

셸이 해석할 때 명령어로 인식한다. 따라서 역 따옴표 안의 내용을 실행한다.

```
# date
# echo `date`      (# echo $(date))
# echo `hostname`  (# echo $(hostname))
# echo "`hostname` is my hostname."

# touch server_`date +%m%d`.log
# tar cvzf /backup/home_`date +%m%d`.tar.gz /home
# while true
> do
> echo "_____`date`_____"

> sleep 2
> done
```

■ `\w` (역 슬래쉬(Back slash))

바로 이후에 있는 문자를 셸이 해석 할 수 없도록 막아 준다.

```
# echo $HOME
# echo \w$HOME      (# echo '$HOME')

# find / -name core -type f -exec rm -f {} \w;
# find / \w( -perm -4000 -o -perm -2000 \w) -type f
```

[참고] `\w`(역 슬래쉬)

```
# \wCMD
# CMD\w
```

```
# ./configure --prefix=/usr/local/apache2 --options2=.... --options3=.... .....
# ./configure --prefix=/usr/local/apache2 \w
> --options2=.... \w
> --options3=.... \w
> --options4=....
```

■ `;` (세미콜론(Semicolon))

한개의 라인에 여러개 명령어 수행할 때 사용한다.

```
# ls
# date
# cal
```

```
# ls ; date ; cal
```

```
# alias pps='ps -ef | head -1 ; ps -ef | grep $1'
# EDITOR=/usr/bin/vim ; export EDITOR      (# export EDITOR=/usr/bin/vim)
```

히스토리(History)

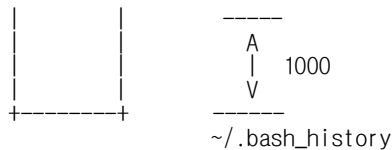
- History 관련 명령어

```
# history
# history -5
# cat ~/.bash_history
# ![history_number]
```
- History 관련 변수

```
HISTSIZE
HISTFILE
HISTFILESIZE
```

6 history CMD

사용자가 로그인 하게 되면 사용자의 명령어를 저장하기 위해서 Stack 공간이 할당된다. Stack 공간은 ksh 셸을 사용하고 있다면, 기본값은 1000개의 명령어를 저장할 수 있는 공간이 할당된다. bash 셸은 1000개의 명령어를 저장할 수 있다. 또한 sh 셸은 히스토리 기능이 없다.



```
HISTSIZE      /* 히스토리를 기록하는 스택의 크기를 지정 */
HISTFILE      /* 히스토리 내용을 지속적으로 저장하는 파일 이름 지정 */
              /* 기본값: ~/.bash_history */
HISTFILESIZE  /* 히스토리파일의 크기 지정 */
```

```
# grep -i histsize /etc/profile
```

HISTSIZE=1000

```
# history | less          /* 현재까지 사용한 스택에 들어간 명령이 출력 */
```

```

1 ifconfig
2 ping 163.126.63.1
3 ping 168.126.63.1
4 vi /etc/sysconfig/network-scripts/ifcfg-eth0
5 vi /etc/sysconfig/network-scripts/ifcfg-eth1
6 vi /etc/hosts
7 vi /etc/resolv.conf
....

```

```
# history -c          /* (clear) 현재까지 사용한 명령을 삭제 */
# history
```

7 alias

별칭(Alias)

```
# alias cp='cp -i'
# alias      (# alias cp)
# unalias cp
```

```
# alias cp='cp -i'
# alias mv='mv -i'
# alias rm='rm -i'
```

```
# alias vi='/usr/bin/vim'
# alias pps='ps -ef | head -1 ; ps -ef | $1'
```

[EX1] 별칭 테스트

```
# alias a='cd /test && rm -rf /test/*'
# alias b='cp /etc/passwd file1 ; cp file1 file2 ; cp file1 file3'
# a
# b
# ls
```


환경 파일(Environmnet File)

- 환경파일의 종류

```
/etc/profile  
~/.bash_profile  
~/.bashrc
```

환경 파일(Environmnet File)

- 사용자 환경 파일 개요
- 사용자 환경파일 읽혀 지는 순서
로그인시 읽혀 지는 환경파일
셸이 실행 될때 마다 읽혀 지는 환경파일
로그아웃 할 때 마다 읽혀 지는 환경파일
- 사용자 환경파일 분석
- 사용자 환경파일 활용

사용자 환경 파일(Bash Initialization)

(1) 환경 파일이 읽혀 지는 순서

(1-1) 로그인시에 읽혀 지는 환경 파일

```
■ /etc/profile
  ■ /etc/profile.d/*.sh
■ ~/.bash_profile (~/.bash_profile -> 파일이 없으면 -> ~/.bash_login -> 파일이 없으면 -> ~/.profile)
  ■ ~/.bashrc
    ■ /etc/bashrc
```

(1-2) 셸이 실행 될때 마다 읽혀 지는 환경 파일

```
■ ~/.bashrc
  ■ /etc/bashrc
    ■ /etc/profile.d/*.sh
```

(1-3) 로그아웃 할 때 마다 읽혀 지는 환경 파일

```
■ ~/.bash_logout
```

(2) 사용자 환경 초기값 설정

(2-1) 관리자가 일반사용자의 환경을 설정 시켜 주는 경우

```
■ /etc/profile      : 로그인 할 때만 읽혀짐
■ /etc/bashrc      : 셸이 실행 될때 마다 읽혀짐
■ /etc/profile.d/*.sh : 셸이 실행 될때 마다 읽혀짐
```

(2-2) 일반 사용자가 자신의 환경을 설정 하는 경우

```
■ ~/.bash_profile  : 로그인 할 때만 읽혀짐
■ ~/.bashrc       : 셸이 실행 될때 마다 읽혀짐
```

(3) 환경파일 분석

(3-1) /etc/profile 파일

cat /etc/profile

```
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

pathmunge () {
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then /* PATH 변수 내용 순서를 설정 */
        if [ "$2" = "after" ] ; then
            PATH=$PATH:$1
        else
            PATH=$1:$PATH
        fi
    fi
}

# ksh workaround
if [ -z "$EUID" -a -x /usr/bin/id ]; then /* EUID 없으면 선언 */
    EUID=`id -u`
    UID=`id -ru`
fi

# Path manipulation
if [ "$EUID" = "0" ]; then /* root 사용자이면 PATH 변수 선언, 위에 존재하는 pathmunge () 사용 */
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /usr/local/sbin
fi

# No core files by default
ulimit -S -c 0 > /dev/null 2>&1 /* -S : Soft 설정, -c : core 파일의 최대 크기 설정 */

if [ -x /usr/bin/id ]; then /* USER, LOGNAME, MAIL 변수 선언 */
    USER=`id -un`
    LOGNAME=$USER
    MAIL="/var/spool/mail/$USER"
fi

HOSTNAME=`/bin/hostname` /* HOSTNAME, HISTSIZE 변수 선언 */
HISTSIZE=1000

if [ -z "$INPUTRC" -a ! -f "$HOME/.inputrc" ]; then
    INPUTRC=/etc/inputrc
fi

export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE INPUTRC

# By default, we want umask to get set. This sets it for login shell
# Current threshold for system reserved uid/gids is 200
# You could check uidgid reservation validity in
# /usr/share/doc/setup-*/uidgid file
if [ $UID -gt 99 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 002
else
    umask 022
fi

for i in /etc/profile.d/*.sh ; do /* /etc/profile.d/*.sh 스크립트 실행 */
    if [ -r "$i" ]; then
        if [ "${-#*i}" != "$-" ]; then
            . $i
        else
            . $i >/dev/null 2>&1
        fi
    fi
done

unset i /* i 변수 unset */
unset pathmunge /* pathmunge 함수 unset */
```

(3-2) /etc/profile.d 디렉토리

```
# cd /etc/profile.d
# ls
```

```
colorls.csh  gnome-ssh-askpass.csh  lang.csh  vim.csh
colorls.sh   gnome-ssh-askpass.sh   lang.sh   vim.sh
glib2.csh    krb5-workstation.csh   less.csh   which-2.sh
glib2.sh     krb5-workstation.sh    less.sh
```

셸(Shell)

- sh style : sh -> ksh -> zsh -> bash
- csh style : csh -> tcsh

(3-3) \$HOME/.bash_profile 파일

```
# cat ~/.bash_profile
```

```
# .bash_profile

# Get the aliases and functions      /* ~/.bashrc 파일 실행 */
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin                /* PATH 변수에 $HOME/bin 추가 */

export PATH                          /* PATH 변수 export */
unset USERNAME                      /* USERNAME 변수 unset */
```

(3-4) \$HOME/.bashrc 파일

```
# cat ~/.bashrc
```

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'                    /* 기본적인 명령어에 대한 alias 선언 */
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions          /* /etc/bashrc 실행 */
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi
```

(3-5) /etc/bashrc 파일

cat /etc/bashrc

```
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# are we an interactive shell?
if [ "$PS1" ]; then
  if [ -z "$PROMPT_COMMAND" ]; then
    case $TERM in
      xterm*)
        if [ -e /etc/sysconfig/bash-prompt-xterm ]; then
          PROMPT_COMMAND=/etc/sysconfig/bash-prompt-xterm
        else
          PROMPT_COMMAND='printf "W033]0;%s@%s:%sW007" "${USER}" "${HOSTNAME%%.*}" W
"${PWD}/${HOME}/~"'
        fi
      ;;
      screen)
        if [ -e /etc/sysconfig/bash-prompt-screen ]; then
          PROMPT_COMMAND=/etc/sysconfig/bash-prompt-screen
        else
          PROMPT_COMMAND='printf "W033]0;%s@%s:%sW033WW" "${USER}" "${HOSTNAME%%.*}" W
"${PWD}/${HOME}/~"'
        fi
      ;;
      *)
        [ -e /etc/sysconfig/bash-prompt-default ] && W
          PROMPT_COMMAND=/etc/sysconfig/bash-prompt-default
        ;;
    esac
  fi
  # Turn on checkwinsize
  shopt -s checkwinsize
  [ "$PS1" = "Wws-WWvWWW$ " ] && PS1="[Wu@Wh WW]WW$ "
fi

if ! shopt -q login_shell ; then
  # We're not a login shell
  # Need to redefine pathmunge, it get's undefined at the end of /etc/profile
  pathmunge () {
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
      if [ "$2" = "after" ] ; then
        PATH=$PATH:$1
      else
        PATH=$1:$PATH
      fi
    fi
  }

  # By default, we want umask to get set. This sets it for non-login shell.
  # You could check uidgid reservation validity in
  # /usr/share/doc/setup-*/uidgid file
  if [ $UID -gt 99 ] && [ "`id -gn`" = "`id -un`" ]; then /* UMASK 값 설정 */
    umask 002
  else
    umask 022
  fi

  # Only display echos from profile.d scripts if we are no login shell
  # and interactive - otherwise just process them to set envvars
  for i in /etc/profile.d/*.sh; do
    if [ -r "$i" ]; then
      if [ "$PS1" ]; then
        . $i
      else
        . $i >/dev/null 2>&1
      fi
    fi
  done

  unset i
  unset pathmunge
fi
# vim:ts=4:sw=4
```

(3-6) \$HOME/.bash_logout

cat ~/.bash_logout

```
# ~/.bash_logout
clear
```

[EX] 사용자 환경 파일이 읽혀 지는 순서에 대한 실습

① /etc/profile 파일에 테스트 내용 추가

■ /etc/profile -> /etc/profile.d/*.sh

cat /etc/profile | head -10

```
echo "|—> /etc/profile read"      <— 새로운 라인 추가

# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

pathmunge () {
    if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
        if [ "$2" = "after" ] ; then
            PATH=$PATH:$1
        else
```

② /etc/profile.d 디렉토리에 테스트용 스크립트 생성

cd /etc/profile.d

cat test.sh

```
#!/bin/bash

echo "|—> /etc/profile.d/*.sh read"
```

③ ~/.bash_profile 파일에 테스트 내용 추가

■ ~/.bash_profile -> ~/.bashrc -> /etc/bashrc

cat ~/.bash_profile | head -10

```
echo "|—> ~/.bash_profile read"    <— 새로운 라인 추가

# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
```

④ ~/.bashrc 파일에 테스트 내용 추가

cat ~/.bashrc | head -10

```
echo "|—> ~/.bashrc read"          <— 새로운 라인 추가

# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then
```

⑤ /etc/bashrc 파일에 테스트 내용 추가
cat /etc/bashrc | head -10

```
echo "|--> /etc/bashrc read"          <---- 새로운 라인 추가

# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# are we an interactive shell?
if [ "$PS1" ]; then
  if [ -z "$PROMPT_COMMAND" ]; then
    case $TERM in
      xterm*)
```

⑥ ~/.bash_logout 파일에 테스트 내용 추가
cat ~/.bash_logout

```
echo "|--> ~/.bash_logout read"      <---- 새로운 라인 추가

# ~/.bash_logout

# clear                                <---- 주석(#) 처리
```

⑦ 사용자로 로그인
telnet localhost

```
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]'.

    linuxXXX.example.com (Linux release 2.6.18-308.el5 #1 SMP Tue Feb 21 20:06:06 EST 2012) (2)

login: root
Password: (centos)
Last login: Thu Aug  1 11:01:49 on :0
You have new mail.
|--> /etc/profile read
|--> /etc/profile.d/*.sh read
|--> ~/.bash_profile read
|--> ~/.bashrc read
|--> /etc/bashrc read
```

⑧ 하위 셸(Sub Shell) 실행
bash

```
|----> ~/.bashrc read
|----> /etc/bashrc read
|----> /etc/profile.d/*.sh read
```

⑨ 새로운 윈도우(Window) 실행
→ 바탕화면(Work Space)에서 오른쪽 마우스 클릭
→ 터미널 열기
→ 출력 내용 확인

```
|----> ~/.bashrc read
|----> /etc/bashrc read
|----> /etc/profile.d/*.sh read
```

exit

⑩ 이전 명령어 윈도우에서 로그 아웃(Logout)

exit
exit

```
echo "|--> ~/.bash_logout read"
```

(4) 환경 파일의 활용 예

(4-1) 사용자 환경 파일에 등록 될수 있는 내용들

- 변수 설정
PATH, PS1, 사용자 정의 변수 설정
- 엘리어스(Alias) 설정
alias ls='ls -hF'
- 셸 자체의 기능

(4-2) 선언 예

cat ~/.bashrc

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'          /* 기본적인 명령어에 대한 alias 선언 */
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions /* /etc/bashrc 실행 */
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

----- ~/.bashrc 파일 하단에 추가 -----

# (a). 기본적인 alias
alias c='clear'           /* 명령어를 짧게 사용하기 위한 설정 */
alias h='history'
alias t='/usr/kerberos/bin/telnet 172.16.9.252'
alias d='cd /test && rm -rf /test/*'

alias grep='/bin/grep -i --color' /* 명령어 옵션 */
alias cat='/bin/cat -n'
alias df='/bin/df -h -T'
alias ls='ls -h --color=tty'

# (b). 필요한 경우 선언하는 alias
alias lsf='/bin/ls -al | grep ^-' /* 새로운 명령어를 위한 alias */
alias lsd='/bin/ls -al | grep ^d'
alias pps='/bin/ps -ef | head -1 ; ps -ef | grep $1'
alias ddf='/bin/df -h -F ext3 ; echo ; /bin/df -i'

alias dir='/bin/ls -ailhF | more'  -----> 윈도우즈 명령어 실행 alias
alias ipconfig='ifconfig'         -----> 윈도우즈 명령어 실행 alias
alias topas='top'                 -----> AIX
alias bdf='df -h'                 -----> HP-UX
alias prstat='top'               -----> Solaris

alias nslookup='/usr/local/bin/nslookup' /* 다른 위치의 명령어 실행 */
alias vi='/usr/bin/vim'

# (c). 편리한 기능의 alias
/*
    Webserver (Apache)
    /etc/httpd/conf/httpd.conf -----> Configuration File
    /var/www/html                  -----> Source Directory
    /etc/httpd/logs                 -----> Log Directory
*/

# 설정 파일
alias fconf='vi /etc/vsftpd/vsftpd.conf'
alias wconf='vi /etc/httpd/conf/httpd.conf'
alias vsftpd.conf='vi /etc/vsftpd/vsftpd.conf'
alias httpd.conf='vi /etc/httpd/conf/httpd.conf'

# 소스디렉토리 이동
alias wdir='cd /var/www/html'

# 로그파일 모니터링
alias mlog='tail -f /var/log/messages'
alias wlog='tail -f /etc/httpd/logs/access_log'
alias welog='tail -f /etc/httpd/logs/error_log'
alias slog='tail -f /var/log/secure'
/* 실무에서 많이 사용되는 로그 파일 이름 형식 */ (예) file_1210.log
# tail -f /logs/file_1210.log
# tail -f /logs/file_`date +%m%d`.log
# alias slog='tail -f /logs/file_`date +%m%d`.log'
# alias slog='view /logs/file_`date --date '1 days ago' +%m%d`.log'
# alias slog='view /logs/file_`date --date '2 days ago' +%m%d`.log'
----- ~/.bashrc 파일 하단에 추가 -----
```