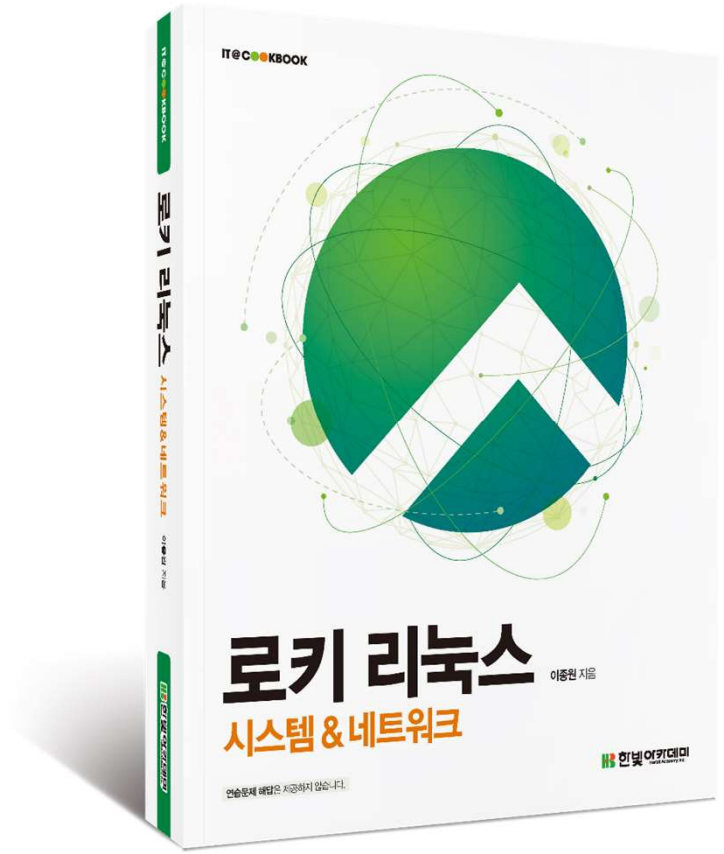
한빛아카데미
Hanbit Academy, Inc.



Chapter 05 셸 사용법

목차

학습목표

- 셸의 기능과 종류를 설명할 수 있다.
- 셸 특수문자의 종류를 이해하고 필요에 따라 적절하게 사용할 수 있다.
- 표준 입출력 장치를 이해하고 입출력 방향을 설정할 수 있다.
- 새로운 앨리어스를 만들거나 필요 없는 앨리어스를 삭제할 수 있다.
- 이스케이프 문자를 이해하고 프롬프트를 원하는 형태로 바꿀 수 있다.
- 시스템과 사용자 환경 설정 파일을 구분하고 사용자 환경을 직접 설정할 수 있다

00 Preview

00 Preview

5장의 내용 구성

- 셸은 리눅스 커널과 사용자를 연결하는 중요한 역할을 수행함
- 사용자가 입력한 내용을 해석하여 관련 명령을 실행하고 그 결과를 출력하므로 사용자는 셸을 통해 리눅스를 만난다고 말할 수 있음
- 배시셸은 유닉스의 전통적인 셸인 본셸과 호환성을 유지하면서 편리한 기능을 제공

01 셀의 기능과 종류

01 셸의 기능과 종류

셸의 기능

- 셸은 사용자와 리눅스 커널 사이에서 중재자 역할을 수행
- 명령어 해석기 기능, 프로그래밍 기능, 사용자 환경 설정 기능으로 구분

명령어 해석기 기능

- 셸이 제공하는 가장 기본적인 기능은 사용자와 커널 사이에서 명령을 해석하여 전달하는 해석기와 번역기 기능
- 셸은 사용자가 입력한 명령이나 파일에서 읽어 들인 명령을 해석하고 적절한 프로그램을 실행
- 로그인 셸: 로그인할 때 실행되는 셸. 사용자가 로그인하면 셸이 자동으로 실행되어 사용자가 명령을 입력하기를 기다림. /etc/passwd 파일에 사용자별로 지정되어 있음

01 셸의 기능과 종류

■ 명령어 해석기 기능

- 프롬프트: 셸이 사용자의 명령을 기다리고 있음을 나타내는 표시
- 셸은 대기 상태로 있다가 사용자가 입력한 명령이나 스크립트를 읽어 들여 우선 이 명령이 셸 자체의 내장 명령인지 판단
- 내장 명령은 특별한 추가 작업 없이 바로 실행
- 내장 명령이 아니라면 별도의 명령 파일이 있다는 뜻이므로, 셸은 자식 프로세스를 만들고 이 자식 프로세스가 명령 파일을 실행하게 함
- 자식 프로세스가 명령 파일을 실행하는 동안 부모인 셸은 대기 상태가 되며, 자식 프로세스가 실행을 종료하면 셸의 대기 상태가 해제되어 복귀하면서 다시 프롬프트를 출력

01 셸의 기능과 종류

프로그래밍 기능

- 셸은 자체 내에 프로그래밍 기능이 있어서 프로그램도 작성할 수 있음
- 셸 스크립트: 셸의 프로그래밍 기능을 사용해 여러 명령을 반복적으로 수행하는 작업으로 만들어진 하나의 셸 프로그램

사용자 환경 설정 기능

- 셸은 사용자 환경을 설정할 수 있도록 초기화 파일 기능을 제공
- 초기화 파일을 사용하면 명령을 찾아오는 경로 설정, 새로 생성되는 파일이나 디렉터리에 대한 기본 권한 설정, 다양한 환경 변수 설정 등을 할 수 있음
- 사용자가 로그인할 때 초기화 파일이 실행되어 사용자 별로 특성에 맞게 초기 환경이 설정됨

01 셸의 기능과 종류

■ 본셸

- 초기에 단순하고 처리 속도가 빨라서 많이 사용되었고, 지금도 시스템 관리 작업을 수행하는 많은 셸 스크립트는 본셸을 기반으로 함
- 명령 이름은 sh
- 사용자의 편의를 위한 기능을 제공하지 않아 사용하기 불편할 수 있음
- 본셸의 경로를 확인해 보면 배시셸과 심볼릭 링크로 연결되어 있음을 알 수 있음

```
[user1@localhost ~]$ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4  1월 24  2023 /bin/sh     → bash
```

01 셸의 기능과 종류

C셸

- 본셸의 기능을 확장한 것으로, 본셸에는 없던 사용자 편의 기능이 들어 있음
- 셸 스크립트 작성을 위한 구문 형식이 C 언어와 같아 C셸이라는 이름이 붙었음
- 명령 이름은 csh

콘셸

- C셸과 달리 본셸과 호환성을 유지할 뿐 아니라 C셸의 특징을 모두 제공하면서 처리 속도도 빠르다는 장점이 있음
- 명령 이름은 ksh

01 셸의 기능과 종류

■ 배시셸

- 본셸과 호환성을 유지하면서 C셸, 콘셸의 편리한 기능도 포함하고 있음
- 명령 이름은 bash
- GPL을 따르는 공개 소프트웨어로서 누구나 자유롭게 사용할 수 있음
- 리눅스의 기본 셸로 제공되고 있어 리눅스 셸로도 많이 알려져 있으며, 로키 리눅스도 배시셸을 기본 셸로 사용하고 있음
- 배시셸의 명령인 bash는 Bourne Again Shell의 약자. 본셸을 업그레이드한 것임을 이름에서도 표현하고 있음

02 셸 기본 사용법

02 셸 기본 사용법

■ 셸 지정 및 변경

- 현재 사용 중인 셸의 종류를 알아보는 가장 쉬운 방법은 시스템에 저장된 사용자 정보를 확인하는 것
- grep 명령을 사용하여 사용자 정보가 들어 있는 파일(/etc/ passwd)에서 특정 사용자의 정보를 찾아 확인하면 됨

```
[user1@localhost ~]$ grep user1 /etc/passwd
user1:x:1000:1000:user1:/home/user1:/bin/bash
```

- 맨 앞에 나온 정보가 로그인 ID이고 맨 마지막에 나온 /bin/bash가 사용자의 기본 셸
- user1 사용자가 로그인할 때 사용자의 기본 셸로 배시셸이 지정되는 것
- 로그인 셸: 사용자가 로그인하면 자동으로 실행되는 기본 셸

02 셸 기본 사용법

■ 기본 셸 변경하기: chsh

- 기본 셸은 배시셸이지만 사용자가 다른 셸을 기본 셸로 사용하고자 한다면 변경할 수 있음
- 로그인 셸은 사용자가 마음대로 바꿀 수 없고 시스템 관리자에게 요청해야 변경할 수 있음
- 리눅스에서는 chsh 라는 명령을 사용하여 사용자가 자신의 로그인 셸을 바꿀 수 있음

chsh

- **기능** 사용자 로그인 셸을 바꾼다.
- **형식** chsh [옵션] [사용자명]
- **옵션** -s shell: 지정하는 셸(절대 경로)로 로그인 셸을 바꾼다.
  -l: /etc/shells 파일에 지정된 셸을 출력한다.
- **사용 예** chsh -l chsh -s /bin/sh user1 chsh

02 셸 기본 사용법

■ 기본 셸 변경하기: chsh

- /etc/shells 파일의 내용을 보면 본셸(/bin/sh)과 배시셸(/bin/bash)이 사용 가능하다는 것을 알 수 있음

```
[user1@localhost ~]$ cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
```

- /etc/shells 파일의 내용은 chsh -l 명령으로도 확인할 수 있음

```
[user1@localhost ~]$ chsh -l
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
```

02 셸 기본 사용법

■ 기본 셸 변경하기: chsh

- 로그인 셸을 본셸로 바꾸려고 시도하는 예
- 단순히 셸 이름만 지정하면 셸의 절대 경로를 입력해야 한다는 메시지가 출력

```
[user1@localhost ~]$ chsh -s sh user1
Changing shell for user1.
chsh: shell must be a full path name        → 절대 경로로 입력하라는 메시지가 출력된다.
```

- 바꾸려는 셸의 절대 경로를 지정하면 셸이 변경 됨

```
[user1@localhost ~]$ chsh -s /bin/sh user1
Changing shell for user1.
암호:                          → 암호에는 user1 계정의 암호를 입력한다.
Shell changed.                 → 로그인 셸이 변경되었다.
```

- /etc/passwd 파일을 확인해 보면 로그인 셸이 /bin/sh로 바뀐 것을 확인할 수 있음

```
[user1@localhost ~]$ grep user1 /etc/passwd
user1:x:1000:1000:user1:/home/user1:/bin/sh
```

02 셸 기본 사용법

■ 로그인 셸과 서브 셸

- 사용자는 프롬프트에서 다른 셸을 실행할 수 있는데 이렇게 새로 생성된 셸을 서브 셸이라고 부름
- 서브 셸은 또 다른 서브 셸을 생성할 수 있으며, 여러 개의 셸이 사슬처럼 연결될 수 있음
- 서브 셸을 종료하는 명령은 ^d(Ctrl +d), exit 등으로 일반 셸의 로그아웃 명령과 같음
- 서브 셸이 종료되면 서브 셸을 실행했던 이전 셸 환경으로 돌아가며, 로그인 셸에서 로그아웃하면 접속이 해제됨

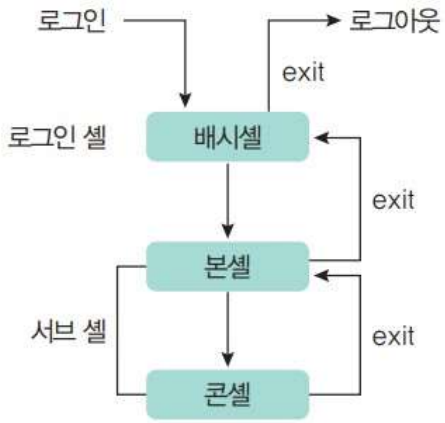


그림 5-1 로그인 셸과 서브 셸

02 셸 기본 사용법

■ 셸 내장 명령

- 셸 내장 명령은 별도의 실행 파일이 없고 셸 안에 자체적으로 포함되어 있음
- 대표적인 셸 내장 명령은 cd
- file 명령으로 /usr/bin/cd 파일의 종류를 확인해 보기. /usr/bin/cd는 셸 스크립트임을 확인할 수 있음

```
[user1@localhost ~]$ file /usr/bin/cd
/usr/bin/cd: a /usr/bin/sh script, ASCII text executable
```

- 셸 스크립트는 텍스트 파일이므로 파일의 내용을 볼 수 있음
- cd는 builtin, 셸 내장 명령임을 확인할 수 있음

```
[user1@localhost ~]$ cat /usr/bin/cd
#!/usr/bin/sh
builtin cd "$@"
```

02 셸 기본 사용법

■ 셸 내장 명령

- 일반적인 실행 파일을 file 명령으로 확인하면 어떻게 출력되는지 확인해 보기

```
[user1@localhost ~]$ file /usr/bin/pwd
/usr/bin/pwd: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV),
(생략)
```

- /usr/bin/pwd 명령을 실행하면 executable로 출력됨
- 실행 파일은 바이너리 파일이므로 cat 명령으로 파일의 내용을 확인할 수 없으니 주의

02 셸 기본 사용법

■ 출력 명령

- 출력 명령은 문자열 출력이나 셸 변수 값 출력, 명령 결과 출력 등에 사용됨
- echo 명령은 모든 셸에서 공통으로 제공하는 출력 명령

echo

- 기능 화면에 한 줄의 문자열을 출력한다.
- 형식 echo [옵션] [문자열]
- 옵션 -n: 마지막에 줄 바꿈을 하지 않는다.
- 사용 예 echo echo text echo -n text

- echo 명령을 사용하여 문자열을 출력하는 예

```
[user1@localhost ~]$ echo linux
linux
[user1@localhost ~]$ echo "Rocky Linux"
Rocky Linux
```

02 셸 기본 사용법

■ 출력 명령

printf

- 기능 자료를 형식화하여 화면에 출력한다.
- 형식 printf [출력 형식] [인수]
- 출력 형식 %d, \n 등 C 언어의 printf 함수의 형식을 지정한다.
- 사용 예 printf text printf "text\n" printf "%d\n" 100

- printf는 % 지시자와 ₩ 문자를 이용하여 출력 형식을 지정할 수 있음

```
[user1@localhost ~]$ printf "Rocky Linux\n"
Rocky Linux
[user1@localhost ~]$ printf "%d + %d = %d\n" 10 10 20
10 + 10 = 20
```

02 셸 기본 사용법

■ 특수문자 사용 방법

- 사용자가 더욱 편리하게 명령을 입력하고 실행할 수 있도록 다양한 특수문자를 제공
- 주요 특수문자로는 *, ?, |, ;, [ ], ~, ' ', " ", ` ` 등이 있음
- 사용자가 명령을 입력하면 셸은 먼저 입력한 내용 중에 특수문자가 있는지 확인하고 이를 해독하여 적절한 형태로 변경한 후 명령을 실행

02 셸 기본 사용법

■ 특수문자 *

- *(별표)는 임의의 문자열을 나타내는 특수문자
- 셸에서 *는 0개 이상의 문자로 대체되며, 주로 여러 파일의 이름을 나열할 때 파일 이름을 간단히 표시하는데 사용
- 명령을 실행할 때 파일명을 적어야 하는 자리에 *를 사용하면 모든 파일을 나타냄

표 5-1 특수문자 *

사용 예	의미
ls *	현재 디렉터리의 모든 파일과 서브 디렉터리를 나열한다. 서브 디렉터리의 내용도 출력한다.
cp * /tmp	현재 디렉터리의 모든 파일을 /tmp 디렉터리 아래로 복사한다.
ls -F t *	t, tmp, temp와 같이 파일명이 t로 시작되는 모든 파일의 이름과 파일 종류를 출력한다. t도 해당한다는 데 주의한다.
cp * .txt ../ch3	확장자가 txt인 모든 파일을 상위 디렉터리 아래의 ch3 디렉터리로 복사한다.
ls -l h * d	파일명이 h로 시작되고 d로 끝나는 모든 파일의 상세 정보를 출력한다. hd, had, hard, h12345d 등 이 조건에 맞는 모든 파일의 정보를 볼 수 있다.

02 셸 기본 사용법

■ 특수문자 ?와 [ ]

- ?(물음표)와 [ ](대괄호)는 모두 하나의 문자를 나타내는 데 사용됨
- ?는 길이가 1인 임의의 한 문자
- [ ]는 괄호 안에 포함된 문자 중 하나. 사용할 여러 문자를 나열하거나 범위를 지정하는 데 사용하며, 다른 특수문자와 혼합하여 사용할 수도 있음

표 5-2 특수문자 ?와 [ ]

사용 예	의미
ls t?.txt	t 다음에 임의의 한 문자가 오고 파일의 확장자가 txt인 모든 파일의 이름을 출력한다. t1.txt, t2.txt, ta.txt 등이 해당된다. 단, t.txt는 제외한다.
ls –l tmp[135].txt	tmp 다음에 1, 3, 5 중 하나가 오고 파일의 확장자가 txt인 모든 파일의 이름을 출력한다. tmp1.txt, tmp3.txt, tmp5.txt 파일이 있으면 해당 파일의 상세 정보를 출력한다. 단, tmp.txt는 제외한다.
ls –l tmp[1–3].txt	[1–3]은 1부터 3까지의 범위를 의미한다. 따라서 ls –l tmp[123].txt와 결과가 같다. 즉 tmp1.txt, tmp2.txt, tmp3.txt 파일이 있으면 해당 파일의 상세 정보를 출력한다.
ls [0–9]*	파일명이 숫자로 시작되는 모든 파일의 목록을 출력한다.
ls [A–Za–z]*[0–9]	파일명이 영문자로 시작되고 숫자로 끝나는 모든 파일의 목록을 출력한다.

02 셸 기본 사용법

■ 특수문자 ~와 -

- ~(물결표)와 -(하이픈)은 디렉터리를 나타내는 특수문자
- ~만 사용하면 현재 작업 중인 사용자의 홈 디렉터리를 나타냄
- 다른 사용자의 로그인 ID와 함께 사용하면(~로그인 ID) 해당 사용자의 홈 디렉터리를 나타냄
- -은 cd 명령으로 현재 디렉터리로 이동하기 직전의 디렉터리를 나타냄

표 5-3 특수문자 ~와 -

사용 예	의미
cp *.txt ~/tmp	확장자가 txt인 모든 파일을 현재 작업 중인 사용자의 홈 디렉터리 아래 tpm 디렉터리로 복사한다.
cp ~user2/ linux.txt .	user2라는 사용자의 홈 디렉터리 아래에서 linux.txt 파일을 찾아 현재 디렉터리로 복사한다.
cd –	이전 작업 디렉터리로 이동한다.

02 셸 기본 사용법

■ 특수문자 ;과 |

- ;(세미콜론)과 |(파이프)는 명령과 명령을 연결
- ;은 연결된 명령을 왼쪽부터 차례로 실행
- |는 왼쪽 명령의 실행 결과를 오른쪽 명령의 입력으로 전달

표 5-4 특수문자 ;과 |

사용 예	의미
date; ls; pwd	왼쪽부터 차례대로 명령을 실행한다. 즉 날짜를 출력한 후 현재 디렉터리의 파일 목록을 출력하고, 마지막으로 현재 작업 디렉터리의 절대 경로를 보여준다.
ls –al / | more	루트 디렉터리에 있는 모든 파일의 상세 정보를 한 화면씩 출력한다. ls –al / 명령의 결과가 more 명령의 입력으로 전달되어 페이지 단위로 출력되는 것이다.

02 셸 기본 사용법

■ 특수문자 ' '와 " "

- ' '(작은따옴표)와 " "(큰따옴표)는 문자를 감싸 문자열로 만들고, 문자열 안의 특수문자 기능을 없앰
- ' '는 모든 특수문자를, " " 는 $, `, ₩를 제외한 모든 특수문자를 일반 문자로 간주하여 처리
- 셸 변수에 저장된 값을 출력할 때 셸 변수 앞에 $를 붙임

표 5-5 특수문자 ' '와 " "

사용 예	의미
echo '$SHELL'	$SHELL 문자열이 화면에 출력된다.
echo "$SHELL"	셸 환경 변수인 SHELL에 저장된 값인 현재 셸의 종류가 화면에 출력된다. /bin/bash를 예로 들 수 있다.

02 셸 기본 사용법

■ 특수문자 ``

- ``(백쿼터)는 키보드 상단의 숫자 1 키 앞에 있는 문자를 사용
- 셸은 ``로 감싼 문자열을 명령으로 해석하여 명령의 실행 결과로 출력

표 5-6 특수문자 ``

사용 예	의미
echo "Today is `date`"	`date`가 명령으로 해석되어 date 명령의 실행 결과로 바뀐다. 결과적으로 다음 예와 같이 출력된다.
	Today is 2023. 08. 20. (일) 17:25:30 KST
ls /usr/bin/`uname -m`	uname -m 명령의 실행 결과를 문자열로 바꾸어 파일 이름으로 사용한다.

02 셸 기본 사용법

■ 특수문자 \

- \ (역슬래시, ₩와 동일함)는 특수문자 바로 앞에 사용. 해당 특수문자의 효과를 없애고 일반 문자처럼 처리

표 5-7 특수문자 \

사용 예	의미
ls –l t*	t*라는 이름을 가진 파일의 상세 정보를 출력한다. \ 없이 t*를 사용하면 t로 시작하는 모든 파일의 상세 정보를 출력한다.
echo \$SHELL	$SHELL을 화면에 출력한다. echo '$SHELL'과 결과가 같다.

■ 특수문자 >, <, >>

- >, >는 입출력의 방향을 바꾸는 특수문자

표 5-8 특수문자 〉, 〈, 〉〉

사용 예	의미
ls –l 〉res	ls –l 명령의 실행 결과를 화면이 아닌 res 파일에 저장한다.
ls –l 〉〉res	ls –l 명령의 실행 결과를 res 파일의 끝부분에 추가한다.
cat 〈text	cat 〈text

03 입출력 방향 변경

03 입출력 방향 변경

■ 표준 입출력 장치

- 표준 입력 장치: 셸이 작업을 수행하는 데 필요한 정보를 받아들이는 장치
- 표준 출력 장치: 실행 결과를 내보내는 장치
- 표준 오류 장치: 표준 출력과 별도로 오류 메시지를 내보내는 장치
- 표준 입력 장치는 키보드로 설정되어 있고, 표준 출력 및 표준 오류 장치는 화면으로 설정되어 있음

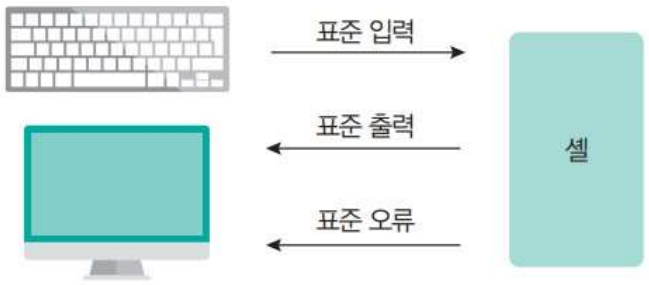


그림 5-2 표준 입출력 장치

03 입출력 방향 변경

■ 표준 입출력 장치

- 파일 디스크립터: 작업 중 필요한 파일에 일련번호를 붙여서 관리하는 값
- 입출력 장치를 변경할 때 파일 디스크립터를 사용

표 5-9 표준 입출력 장치의 파일 디스크립터

파일 디스크립터	파일 디스크립터 대신 사용하는 이름	정의
0	stdin	명령의 표준 입력
1	stdout	명령의 표준 출력
2	stderr	명령의 표준 오류

03 입출력 방향 변경

출력 리다이렉션

- 리다이렉션: 표준 입출력 장치를 파일로 바꾸는 것
- 출력 결과를 저장할 파일이 이미 존재하는 파일일 경우, 기존 파일의 내용을 유지할지 말지에 따라 달라짐
- 기존 파일의 내용을 삭제하고 새로 결과를 저장할 때는 >를, 기존 파일의 내용 뒤에 결과를 추가할 때는 >>를 사용

03 입출력 방향 변경

■ 파일 덮어쓰기: >

>

- 기능 파일 리다이렉션(덮어쓰기)을 한다.
- 형식 명령 1> 파일명
  명령 > 파일명

- 표준 출력 파일을 바꾸는 특수문자 >
- 첫 번째 형식의 1은 파일 디스크립터 1번을 의미
- 파일 디스크립터 1은 생략 가능하며, 보통 1이 생략된 두 번째 형식을 사용
- 셸은 >를 사용한 리다이렉션에서 지정한 파일명의 파일이 없으면 파일을 생성하여 명령의 수행 결과를 저장
- 해당 파일이 있으면 기존 내용이 없어지고 명령의 수행 결과로 대체되므로 출력 리다이렉션을 사용할 때는 먼저 해당 이름의 파일이 있는지 확인해야 함

03 입출력 방향 변경

■ 파일 덮어쓰기: >

- ls -al 명령의 결과를 out1로 리다이렉션했다가 다시 date 명령의 결과를 out1로 리다이렉션 했음
- 두 번 모두 특수문자 >를 사용했으므로 파일 내용을 덮어씀
- 최종적으로 수행한 리다이렉션만 파일에 남아 date 명령의 결과가 저장된 것

```
[user1@localhost ~]$ mkdir Test/ch5
[user1@localhost ~]$ cd Test/ch5
[user1@localhost ch5]$ ls out1              → out1이 있는지 확인한다.
ls: cannot access 'out1': 그런 파일이나 디렉터리가 없습니다
[user1@localhost ch5]$ ls -al               → 명령의 결과가 화면(표준 출력)으로 출력된다.
합계 0
drwxr-xr-x. 2 user1 user1  6  8월 20 18:44 .
drwxr-xr-x. 6 user1 user1 50  8월 20 18:44 ..
[user1@localhost ch5]$ ls -al > out         → 명령의 결과를 out1 파일에 저장한다.
[user1@localhost ch5]$ cat out1             → 파일 내용을 확인한다.
합계 0
drwxr-xr-x. 2 user1 user1 18  8월 20 18:44 .
drwxr-xr-x. 6 user1 user1 50  8월 20 18:44 ..
-rw-r--r--. 1 user1 user1  0  8월 20 18:44 out1
[user1@localhost ch5]$ date > out1          → 명령의 결과를 out1 파일에 저장한다.
[user1@localhost ch5]$ cat out1
2023. 08. 20. (일) 18:44:43 KST             → ls 명령의 실행 결과가 없어졌다.
```

03 입출력 방향 변경

■ 파일 덮어쓰기: >

```
[user1@localhost ch5]$ set -o noclobber
[user1@localhost ch5]$ ls > out1
-bash: out1: 이미 있는 파일을 덮어쓸 수 없음      → 파일에 저장하는 것이 거부되었다.
```

- set 명령을 사용하면 예상치 않게 파일 내용을 덮어쓰는 일을 예방할 수 있음

```
[user1@localhost ch5]$ set +o noclobber
[user1@localhost ch5]$ ls > out1                  → 파일에 저장되었다.
```

- 설정을 해제하려면 다음과 같이 + 옵션을 사용

03 입출력 방향 변경

■ 파일 덮어쓰기: >

- cat 명령은 파일 내용을 화면에 출력할 때 사용
- cat 명령은 표준 입력으로부터 입력을 받아 표준 출력으로 보낼 수도 있음
- cat 명령의 결과를 파일로 리다이렉션하면 키보드의 입력을 새로운 파일로 저장할 수 있음

```
[user1@localhost ch5]$ cat > out1   → 표준 입력을 받아 out1에 저장한다.
Rocky Linux                         → 내용을 입력한다.
I love Linux.
^D                                  → Ctrl+d, 입력을 종료한다.
[user1@localhost ch5]$ cat out1     → 파일 내용을 확인한다.
Rocky Linux
I love Linux.
```

03 입출력 방향 변경

■ 파일에 내용 추가하기: >>

- 표준 출력을 리다이렉션하는 두 번째 특수문자. 명령의 실행 결과를 지정한 파일 의 끝에 덧붙임

»

- 기능 파일에 내용을 추가한다.
- 형식 명령 » 파일명

- >>를 사용한 출력 리다이렉션에서는 지정한 파일명의 파일이 없으면 파일을 생성하고, 해당 파일이 있으면 그 파일의 끝에 명령의 실행 결과를 추가함

```
[user1@localhost ch5]$ cat out1      → 기존 파일 내용을 확인한다.
Rocky Linux
I love Linux.
[user1@localhost ch5]$ date » out1   → 출력 리다이렉션한다(내용 추가).
[user1@localhost ch5]$ cat out1      → 파일 내용을 확인한다.
Rocky Linux
I love Linux.
2023. 08. 20. (일) 19:16:05 KST      → 내용이 추가되어 있다.
```

03 입출력 방향 변경

■ 오류 리다이렉션

- 명령을 실행하는 과정에서 셸이 화면에 출력하는 메시지는 명령의 정상적인 실행 결과와 명령 실행 중 발생하는 오류 메시지 두 가지임
- 정상적인 실행 결과는 표준 출력이고, 오류 메시지는 표준 오류라고 함
- 표준 오류도 기본적으로 화면에 출력되며 표준 출력처럼 리다이렉션할 수 있음

```
[user1@localhost ch5]$ ls
out1                                                    → 정상 실행(표준 출력)
[user1@localhost ch5]$ ls /abc
ls: cannot access '/abc': 그런 파일이나 디렉터리가 없습니다        → 오류 메시지(표준 오류)
```

- 첫 번째 명령은 정상적으로 처리되어 표준 출력 장치로 출력되었고, 두 번째는 정상적으로 처리되지 않았기 때문에 오류메시지가 표준 오류 장치로 출력 됨
- 표준 출력과 표준 오류는 같은 것처럼 보이지만 다르게 관리되고 있음

03 입출력 방향 변경

■ 오류 리다이렉션

```
[user1@localhost ch5]$ ls > ls.out                              → 표준 출력 리다이렉션
[user1@localhost ch5]$ ls /abc > ls.err                         → 표준 출력 리다이렉션
ls: cannot access '/abc': 그런 파일이나 디렉터리가 없습니다.   → 오류 메시지가 화면에 출력된다.
[user1@localhost ch5]$ cat ls.err                               → 오류 메시지가 저장되지 않았다.
[user1@localhost ch5]$ cat ls.out                               → 표준 출력 내용이 출력된다.
ls.out
out1
```

- 오류 메시지는 표준 출력이 아니기 때문에 리다이렉션되지 않음
- 표준 출력은 1번, 표준 오류는 2번 장치 파일
- >만으로는 표준 오류가 리다이렉션되지 않음. >는 1>와 같은 것으로, 결국 1번 파일 대신 사용한 파일명을 지정하는 것이기 때문

03 입출력 방향 변경

■ 오류 리다이렉션

2>

- **기능** 표준 오류 메시지를 파일에 저장한다.
- **형식** 명령 2> 파일명

- 2>는 2번 파일, 즉 stderr 파일로 나갈 내용을 지정한 파일에 저장하라고 셸에 알려주는 것
- 오류 리다이렉션에서는 파일 디스크립터 번호를 생략할 수 없음

```
[user1@localhost ch5]$ ls /abc 2> ls.err                    → 표준 오류를 리다이렉션한다.
[user1@localhost ch5]$ cat ls.err
ls: cannot access '/abc': 그런 파일이나 디렉터리가 없습니다 → 파일에 오류 메시지가 저장되어 있다.
```

03 입출력 방향 변경

■ 표준 출력과 표준 오류를 한 번에 리다이렉션하기

- 명령의 정상적인 실행 결과와 오류 메시지를 한 번에 리다이렉션하여 각각 파일에 저장하려 면 >와 2>를 함께 사용하면 됨

```
[user1@localhost ch5]$ ls . /abc > ls.out 2> ls.err
```

■ 오류 메시지 버리기

- 명령 실행 중 나타나는 오류 메시지를 무시하고 싶을 때는 리눅스의 특수 파일인 /dev/null 을 사용하면 됨
- /dev/null 파일은 버리고 싶은 내용을 담는 장치 파일로, 이 파일로 전달 된 모든 내용은 시스템에서 사라지고 복구할 수 없음

```
[user1@localhost ch5]$ ls /abc 2> /dev/null
```

03 입출력 방향 변경

■ 표준 출력과 표준 오류를 한 파일로 리다이렉션하기

```
[user1@localhost ch5]$ ls . /abc > ls.out 2>&1
[user1@localhost ch5]$ cat ls.out
ls: cannot access '/abc': 그런 파일이나 디렉터리가 없습니다   → 저장된 오류 메시지다.
.:                                                          → 현재 디렉터리 내용이다.
ls.err
ls.out
out1
```

① 명령의 정상 실행 결과를 파일로 리다이렉션한다(>)

② 그 명령 전체의 오류 메시지를 1번 파일(표준 출력 파일, &1이라고 표현함)로 리다이렉션 한다(2>)

- ①번을 실행하면서 표준 출력이 먼저 변경됨. 표준 출력이 변경된 다음이므로 ②번에서는 오류 메시지를 1번 파일로 보내기만 하면 됨
- ②는 2>&1과 같이 지정함. 이는 2번 파일을 1번으로 리다이렉션하라는 뜻. 이때 2>와 &1 사이에 공백이 있으면 안 됨

03 입출력 방향 변경

■ 입력 리다이렉션

- <는 표준 입력 장치 파일을 리다이렉션하는 기능을 제공하는 특수문자

<

- 기능 표준 입력을 바꾼다.
- 형식 명령 0< 파일명
  명령 < 파일명

- cat 명령은 파일의 내용이나 키보드로 입력을 받아 화면에 출력하는 명령
- cat 명령이 파일을 읽어서 출력하는 기능은 <를 생략한 것으로, 표준 입력을 자동으로 파일로 리다이렉션한 것

03 입출력 방향 변경

■ 입력 리다이렉션

- 비교해 보기

```
[user1@localhost ch5]$ cat out1              → 파일 내용을 출력한다(< 생략).
Rocky Linux
I love Linux.
2023. 08. 20. (일) 19:16:05 KST
[user1@localhost ch5]$ cat < out1            → 표준 입력을 리다이렉션한다(< 사용).
Rocky Linux
I love Linux.
2023. 08. 20. (일) 19:16:05 KST
[user1@localhost ch5]$ cat 0< out1           → 표준 입력을 리다이렉션한다(0< 사용).
Rocky Linux
I love Linux.
2023. 08. 20. (일) 19:16:05 KST
```

03 입출력 방향 변경

■ 특수문자 사용하고 입출력 방향 바꾸기

① 현재 디렉터리에서 l(소문자 L)로 시작하는 모든 파일의 상세 정보를 확인

```
[user1@localhost ch5]$ ls -l l*
```

② 임시로 사용할 temp 디렉터리를 만듦

```
[user1@localhost ch5]$ mkdir temp
```

③ l로 시작하는 모든 파일을 temp 디렉터리로 이동

```
[user1@localhost ch5]$ mv l* temp
```

④ uname 명령은 운영체제의 이름을 출력. 이 명령을 사용하여 ‘ This is Linux System.’ 이라는 문장을 출력. 백쿼터 기호를 작은따옴표와 헷갈리지 않도록 유의함

```
[user1@localhost ch5]$ echo "This is `uname` System."
This is Linux System.
```

03 입출력 방향 변경

■ 특수문자 사용하고 입출력 방향 바꾸기

⑤ 출력 방향 바꾸기를 통해 ④번 명령의 실행 결과를 u.out 파일에 저장

```
[user1@localhost ch5]$ echo "This is `uname` System." > u.out
```

⑥ date 명령의 실행 결과를 u.out 파일에 저장. u.out 파일에는 어떤 내용이 남아 있을지 생각해 봄

```
[user1@localhost ch5]$ date > u.out
```

03 입출력 방향 변경

■ 특수문자 사용하고 입출력 방향 바꾸기

혼자해보기 특수문자 사용하고 입출력 방향 바꾸기

❶ 특수문자 *를 사용하여 temp 디렉터리로 이동한 모든 파일을 현재 디렉터리로 복사한다.

❷ uname -n 명령은 호스트 이름을 출력한다. 이 명령과 특수문자 ` `를 사용하여 'This system name is ****.'와 같이 출력되도록 명령을 실행한다. ****에는 실제 호스트 이름이 출력되어야 한다.

❸ 앞의 ❷번에서 실행한 결과를 출력 방향 바꾸기로 un.out 파일에 저장한다.

❹ 현재 디렉터리의 파일 목록을 출력 방향 바꾸기로 ls.out 파일에 저장한다.

❺ un.out, ls.out 파일의 내용을 모두 출력 방향 바꾸기를 사용하여 u.dat 파일에 추가한다. 명령을 한 번에 실행해야 한다.

04 배시셸의 환경 변수

04 배시셸의 환경 변수

배시셸의 환경 변수

- 셸은 셸의 환경을 설정하기 위한 값을 저장할 수 있도록 변수를 제공. 이 변수를 이용하여 사용자마다 각각 원하는 환경을 설정할 수 있음
- 이 변수에는 사용자 정보, 명령을 찾아오는 경로, 각 사용자의 프롬프트 모양 등 다양한 정보가 저장되어 있어 사용자의 작업 환경을 더욱 편리하게 설정할 수 있도록 해 줌

셸 변수와 환경 변수

- 셸이 사용하는 변수에는 셸 변수와 환경 변수가 있음
- 셸 변수는 현재 셸에서만 사용할 수 있고 서브 셸로는 전달되지 않지만, 환경 변수는 현재 셸뿐 아니라 서브 셸로도 전달됨
- 셸 변수는 해당 셸의 지역 변수이고, 환경 변수는 현재 셸과 서브 셸에서 공통으로 사용할 수 있는 전역 변수라고 보면 됨

04 배시셸의 환경 변수

■ 셸 변수와 환경 변수

- 셸 변수는 셸별로 따로 지정되어 서로 독립적인 반면, 환경 변수는 로그인 셸과 서브 셸 모두에 공통으로 적용
- 일반적으로 환경 변수의 이름에는 대문자를 사용함. 소문자로 지정해도 문제는 없지만 관습적으로 대문자를 사용하므로 특별한 이유가 없는 한 따르는 것이 좋음

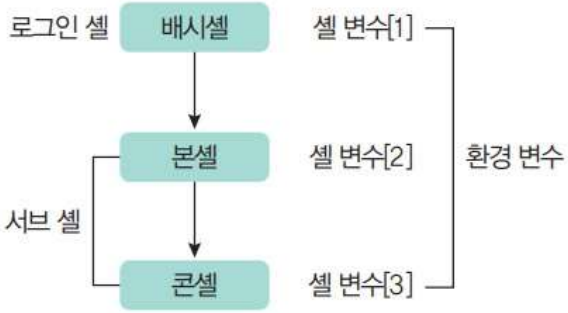


그림 5-3 셸 변수와 환경 변수

표 5-10 주요 셸 환경 변수

환경 변수	의미	환경 변수	의미
HISTSIZE	히스토리 저장 크기	PATH	명령을 탐색할 경로
HOME	사용자 홈 디렉터리의 절대 경로	PWD	작업 디렉터리의 절대 경로
LANG	사용하는 언어	SHELL	로그인 셸
LOGNAME	사용자 계정 이름		

04 배시셸의 환경 변수

■ 전체 변수 출력하기: set, env

- 셸의 변수를 출력하는 명령은 set과 env
- Set은 셸 변수와 환경 변수를 모두 출력하고 env는 환경 변수만 출력
- set 명령의 실행 결과

```
[user1@localhost ch5]$ set | more
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force_
fignore:globasciiran ges:histappend:interactive_comments:login_shell:progcomp:promptvars:
sourcepath
(생략)
which ()
{
    ( alias;
     eval ${which_declare} ) | /usr/bin/which --tty-only --read-alias --read-functions
--show-tilde --show-dot $@
}
```

04 배시셸의 환경 변수

■ 전체 변수 출력하기: set, env

- env 명령의 실행 결과

```
[user1@localhost ch5]$ env
SHELL=/bin/bash
HISTCONTROL=ignoredups
HISTSIZE=1000
HOSTNAME=localhost
PWD=/home/user1/Test/ch5
(생략)
```

04 배시셸의 환경 변수

■ 특정 변수 출력하기: echo

- 전체 변수를 한 번에 출력하지 않고 개별 변수의 값을 출력할 수도 있음
- 변수의 값을 출력할 때는 변수명 앞에 특수문자 $를 붙여야 함
- 사용자의 로그인 셸을 저장하고 있는 환경 변수인 SHELL 변수의 값을 출력한 예

```
[user1@localhost ch5]$ echo $SHELL
/bin/bash
```

04 배시셸의 환경 변수

■ 셸 변수와 환경 변수 설정

- 변수명과 문자열 사이에 공백이 있으면 안 됨

셸 변수 정의

- 형식 변수명=문자열
- 사용 예 SOME=test

- 셸 변수를 설정하는 예

```
[user1@localhost ch5]$ SOME=test
[user1@localhost ch5]$ echo $SOME
test
```

- env 명령으로는 SOME 변수를 확인할 수 없음. SOME이 아직 환경 변수로 설정되지 않았기 때문

```
[user1@localhost ch5]$ set | grep SOME
SOME=test
[user1@localhost ch5]$ env | grep SOME
[user1@localhost ch5]$
```

04 배시셸의 환경 변수

■ 환경 변수 설정하기: export

- 셸 변수를 정의하고, export 명령을 사용하여 이를 환경 변수로 바꿈. export는 셸 내장 명령

export

- 기능 지정한 셸 변수를 환경 변수로 바꾼다.
- 형식 export [옵션] [셸 변수]
- 옵션 -n: 환경 변수를 셸 변수로 변경한다.
- 사용 예 export
  export SOME
  export SOME=test

- SOME을 환경 변수로 변경할 수 있음

```
[user1@localhost ch5]$ export SOME
[user1@localhost ch5]$ env | grep SOME
SOME=test
```

04 배시셸의 환경 변수

■ 환경 변수 설정하기: export

- export 명령과 함께 변수를 정의하면 한 번에 환경 변수로 설정할 수도 있음

```
[user1@localhost ch5]$ export SOME1=test1
[user1@localhost ch5]$ env | grep SOME1
SOME1=test1
```

■ 환경 변수를 다시 셸 변수로 변경하기: export -n

- 환경 변수로 전환한 셸 변수를 다시 셸 변수로 변경하려면 export -n 명령을 사용
- 앞에서 환경 변수로 설정한 SOME1을 다시 셸 변수로 바꾸는 예

```
[user1@localhost ch5]$ export -n SOME1
[user1@localhost ch5]$ env | grep SOME*
SOME=test
```

04 배시셸의 환경 변수

■ 변수 해제

unset

- 기능 지정한 변수를 해제한다.
- 형식 unset [변수]
- 사용 예 unset SOME

- SOME, SOME1 변수를 해제해 보기

```
[user1@localhost ch5]$ unset SOME
[user1@localhost ch5]$ unset SOME1
[user1@localhost ch5]$ echo $SOME

[user1@localhost ch5]$ echo $SOME1
```

- 변수 설정이 해제되었으므로 아무 값도 출력되지 않음

05 앨리어스와 히스토리

05 앨리어스와 히스토리

■ 앨리어스

- 앨리어스: 우리말로 '별명'을 뜻함. 기존 명령을 대신하여 다른 이름(별칭)을 붙일 수 있도록 하는 기능
- 긴 명령 대신 짧은 명령을 만들어 사용할 수 있고, 여러 명령을 연결하여 하나의 명령으로 만들 수도 있으며, 자주 사용하는 옵션을 포함하여 새로운 이름을 붙여서 사용할 수도 있음

alias

- 기능 앨리어스를 생성한다.
- 형식 alias 이름='명령'
- 사용 예 alias: 현재 설정된 별칭 목록을 출력한다.
  alias 이름='명령': 명령을 수정해 사용하는 경우다.
  alias 이름='명령;명령2;...': 여러 명령을 하나의 이름으로 사용하는 경우다.

05 앨리어스와 히스토리

■ 앨리어스

- 아무것도 지정하지 않고 alias 명령을 실행시키면 현재 설정된 앨리어스가 출력 됨. 기본으로 설정된 앨리어스는 시스템에 따라 다름

```
[user1@localhost ch5]$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto '
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
(생략)
```

05 앨리어스와 히스토리

■ 앨리어스

```
[user1@localhost ch5]$ ls
out1  temp  u.out
[user1@localhost ch5]$ l.
.  ..
[user1@localhost ch5]$ ll
합계 8
-rw-r--r--. 1 user1 user1 59   8월 20 19:16 out1
drwxr-xr-x. 2 user1 user1 34   8월 20 20:05 temp
-rw-r--r--. 1 user1 user1 33   8월 20 20:06 u.out
```

- l.은 ls -d .*를 앨리어스한 것이며 .(마침표)로 시작하는 경로를 보여줌
- ll은 ls -l 명령을 실행하도록 앨리어스 한 것
- 자주 사용하는 명령과 옵션을 묶어서 앨리어스를 설정하면 편리함

05 앨리어스와 히스토리

■ 앨리어스 설정 및 삭제하기

- 셸 변수를 지정하듯이 '앨리어스 이름=명령' 형식을 사용.
- =를 중심으로 좌우에 공백이 있으면 안 됨. 설정하는 명령에서 옵션 등을 지정하여 공백이 있을 경우에는 작은따옴표로 묶어야 함
- ls 명령에 -F 옵션을 지정하여 다시 ls 명령으로 지정하는 예

```
[user1@localhost ch5]$ ls
out1  temp  u.out
[user1@localhost ch5]$ alias ls='ls -F'      → 공백이 있으면 작은따옴표를 사용한다.
[user1@localhost ch5]$ ls                    → 앨리어스의 ls를 실행한다.
out1  temp/  u.out                           → ls -F의 결과를 출력한다.
```

- 앨리어스에서는 앨리어스 이름으로 기존 명령을 그대로 사용할 수 있음. 사용자가 입력한 명령을 실행할 때 앨리어스가 있는지 셸이 먼저 확인하기 때문

05 앨리어스와 히스토리

■ 앨리어스 설정 및 삭제하기

```
[user1@localhost ch5]$ alias rm='rm -i'
[user1@localhost ch5]$ rm out1
rm: remove 일반 파일 'out1'?
```

- 파일 삭제 명령인 rm으로 파일을 무심코 삭제하면 복구가 불가능. 이를 방지하기 위해 앨리어스를 사용하여 rm 명령에 안전장치를 둘 수 있음
- 이렇게 하면 파일을 삭제할 때마다 -i 옵션으로 삭제 여부를 확인하고 y로 승인된 경우에만 삭제

05 앨리어스와 히스토리

■ 앨리어스 설정 및 삭제하기

unalias

- 기능 앨리어스를 삭제한다.
- 형식 unalias 앨리어스

- 기존 앨리어스를 사용하지 않는다면 unalias 명령을 사용하여 삭제해야 함

```
[user1@localhost ch5]$ unalias ls
[user1@localhost ch5]$ unalias rm
```

05 앨리어스와 히스토리

■ 앨리어스에 인자 전달하기

- cd 명령과 pwd 명령을 묶어서 앨리어스를 만들 수 있음

```
[user1@localhost ch5]$ alias cd='cd;pwd'    → ;으로 명령을 연결하여 앨리어스를 만든다.
[user1@localhost ch5]$ cd
/home/user1
[user1@localhost ~]$ cd Temt/ch5
/home/user1
```

- 앨리어스를 만들어 실행하면 cd 명령을 수행해도 계속 사용자의 홈 디렉터리로만 감. 원하는 디렉터리로 이동하려면 앨리어스를 설정할 때 cd에 인자로 디렉터리명을 지정해야 하는데 그런 인자가 없기 때문
- 배시셸에서는 앨리어스로 인자를 전달할 수 없음

05 앨리어스와 히스토리

■ 앨리어스에 인자 전달하기

- 배시셸에서 인자를 전달하려면 프로그래밍 기능에서 함수를 사용해야 함
- 함수를 입력하기 전에 먼저 unalias 명령으로 기존 앨리어스를 삭제해야 함
- 함수는 기존 명령의 이름을 그대로 사용할 수 없으므로 다른 이름을 사용해야 함

```
[user1@localhost ~]$ unalias cd                  → 앨리어스를 해제한다.
[[user1@localhost ~]$ function cdpwd {           → 함수 입력을 시작한다.
> cd $1;pwd                                      → 프롬프트가 >로 바뀐다. 내용을 입력한다.
> }                                              → 함수 입력을 종료한다.
[user1@localhost ~]$ cdpwd /tmp                  → 함수를 실행해 본다.
/tmp
[user1@localhost tmp]$
```

05 앨리어스와 히스토리

■ 이전 명령 보기: history

- 사용자가 로그인하여 작업한 명령은 임시 버퍼에 저장되는데, 배시셸 내장 명령인 history 명령으로 다시 불러서 사용할 수 있음

history

- 기능 히스토리(명령 입력 기록)를 출력한다.
- 형식 history

```
[user1@localhost tmp]$ history
(생략)
 457  function cdpwd { cd $1;pwd; }
 458  cdpwd /tmp
 459  history
```

05 앨리어스와 히스토리

■ 명령 재실행하기: !

- 히스토리로 저장된 명령을 다시 불러서 재실행할 수 있음

표 5-11 !를 사용한 명령 재실행 방법

사용법	기능
!!	바로 직전에 실행한 명령을 재실행한다.
!번호	히스토리에서 해당 번호의 명령을 재실행한다.
!문자열	히스토리에서 해당 문자열로 시작하는 마지막 명령을 재실행한다.

- !!는 바로 직전에 실행한 명령을 다시 실행

```
[user1@localhost tmp]$ cd ~/Test/ch5
[user1@localhost ch5]$ ls
out1  temp  u.out
[user1@localhost ch5]$ !!       → 바로 직전 명령을 재실행한다.
ls
out1  temp  u.out
```

05 앨리어스와 히스토리

■ 명령 재실행하기: !

- 직전 명령이 아니라 이전에 실행한 명령을 재실행할 때는 먼저 history 명령으로 확인하고 재실행할 명령의 히스토리 번호나 명령의 앞 글자를 이용함

```
[user1@localhost ch5]$ history  → 히스토리를 확인한다.
(생략)
  461  cd ~/Test/ch5
  462  ls
  463  history
[user1@localhost ch5]$ !462     → 히스토리 번호로 재실행한다.
ls

out1  temp  u.out
[user1@localhost ch5]$ !l       → 명령의 앞 글자로 재실행한다.
ls
out1  temp  u.out
```

- 명령의 앞 글자를 이용할 경우, 해당하는 글자로 시작하는 명령 중에서 가장 마지막에 실행한 명령이 재실행됨

05 앨리어스와 히스토리

■ 명령 편집 및 재실행하기

- 화살표 키를 사용하여 오류가 난 명령을 프롬프트로 불러내서 수정한 뒤 재실행할 수 있음
- ↑ , ↓ 로 이전 명령을 불러온 다음 ← , → 를 사용하여 명령의 잘못된 부분을 수정하면 됨
- 명령에 hisdory라고 잘못 입력

```
[user1@localhost ch5]$ man hisdory
No manual entry for hisdory
```

- 프롬프트에서 ↑ 를 누르면 방금 실행한 명령이 다시 나타나고 커서가 행의 끝에서 대기

```
[user1@localhost ch5]$ man hisdory
```

- 이 상태에서 ← , → 로 커서를 이동시키고 Back Space 로 삭제한 후 다시 글자를 입력하면 됨

```
[user1@localhost ch5]$ man history
```

- Enter 를 입력하면 명령이 재실행됨

05 앨리어스와 히스토리

■ 환경 변수, 앨리어스, 히스토리

① 셸 변수 TESTA를 설정하고 출력해 봄

```
[user1@localhost ch5]$ TESTA=testa
[user1@localhost ch5]$ echo $TESTA
testa
```

② 본셸(sh)을 실행하여 서브 셸로 이동

```
[user1@localhost ch5]$ sh
sh-5.1$
```

③ 셸 변수 TESTA가 출력되는지 확인함. 출력되지 않는 이유는?

```
sh-5.1$ echo $TESTA

sh-5.1$
```

05 앨리어스와 히스토리

■ 환경 변수, 앨리어스, 히스토리

④ 서브 셸에서 로그인 셸로 복귀

```
sh-5.1$ exit
exit
[user1@localhost ch5]$
```

⑤ pwd 명령과 ls 명령을 묶어서 앨리어스 pls를 만듦

```
[user1@localhost ch5]$ alias pls='pwd;ls'
[user1@localhost ch5]$ pls
/home/user1/Test/ch5
out1  temp  u.out
```

05 앨리어스와 히스토리

■ 환경 변수, 앨리어스, 히스토리

⑥ ↑로 이전 명령을 불러서 clear 명령에 대한 앨리어스 c를 만듦

```
[user1@localhost ch5]$ ↑                    → ↑를 누른다.
[user1@localhost ch5]$ pls                  → 바로 이전 명령인 pls를 출력한다.
[user1@localhost ch5]$ ↑                    → 다시 ↑를 누른다.
[user1@localhost ch5]$ alias pls='pwd;ls'   → alias 명령을 출력한다.
[user1@localhost ch5]$ alias c=clear        → ←→를 사용하여 명령을 수정한다.
```

혼자해보기 환경 변수, 앨리어스, 히스토리

❶ 셸 변수 TESTB를 설정하고 환경 변수로 만든다.
❷ 본셸을 실행하여 서브 셸로 이동한다.
❸ TESTA와 TESTB 변수를 출력해 본다. 어떤 변수가 출력되는가? 왜 그럴까?
❹ 서브 셸에서 로그인 셸로 다시 복귀한다.
❺ history 명령을 앨리어스 h로 만든다.
❻ alias 명령을 앨리어스 a로 만든다. ↑로 이전 명령을 불러내서 수정한다.

06 프롬프트 설정

06 프롬프트 설정

■ 이스케이프 문자와 프롬프트 설정

- 배시셸에서는 환경 변수 PS1에 프롬프트로 사용할 문자열을 저장

```
[user1@localhost ch5]$ echo $PS1
[\u@\h \W]\$                    ➞ PS1의 현재 설정값
```

- \로 시작하는 특별한 문자를 이스케이프 문자라고 함. \u와 같이 \로 시작하는 이스케이프 문자들은 두 글자가 아니라 한 글자로 처리됨
- 이스케이프 문자는 화면에 문자 그대로 출력되지 않고 셸이 문자의 의미를 해석하여 실행

06 프롬프트 설정

■ 이스케이프 문자와 프롬프트 설정

- [\u@\h \W]\$를 해석하면 \u는 사용자 이름, \h는 호스트 이름, \W는 현재 작업 디렉터리명
- $ 앞의 \는 $를 특수문자가 아닌 일반 문자로 인식하게 함
- 따라서 [user1@localhost ~]$와 같은 형태가 되는 것

표 5-12 이스케이프 문자

이스케이프 문자	기능
\a	ASCII 종소리 문자(07)
\d	'요일 월 일' 형식으로 날짜를 표시한다(예 Wed May 1).
\e	ASCII의 이스케이프 문자로 터미널에 고급 옵션을 전달한다.
\h	첫 번째 .(마침표)까지의 호스트 이름(예 server.co.kr에서 server)
\H	전체 호스트 이름
\n	줄 바꾸기
\s	셸 이름
\t	24시간 형식으로 현재 시간을 표시한다(HH:MM:SS 형식).
\T	12시간 형식으로 현재 시간을 표시한다(HH:MM:SS 형식).
\@	12시간 형식으로 현재 시간을 표시한다(오전/오후 형식).
\u	사용자 이름
\v	배시셸의 버전
\w	현재 작업 디렉터리(절대 경로)
\W	현재 작업 디렉터리의 절대 경로에서 마지막 디렉터리명
\!	현재 명령의 히스토리 번호
\[	출력하지 않을 문자열의 시작 부분을 표시한다.
\]	출력하지 않을 문자열의 끝부분을 표시한다.

06 프롬프트 설정

■ 이스케이프 문자와 프롬프트 설정

- 프롬프트를 바꾸는 것은 환경 변수 PS1에 새로운 형태의 문자열을 지정하는 것

```
[user1@localhost ch5]$ PROMPT=$PS1
```

■ 프롬프트를 다양한 형태로 바꿔보기

① 간단한 문자열로 바꿔보기. 프롬프트의 끝을 표시하기 위해 마지막에]나 $ 같은 표시를 하고 공백문자를 하나 두는 것이 좋음. 공백이 없는 경우, 프롬프트로 사용하는 문자열과 사용자가 입력한 명령을 구분하기가 어렵기 때문

```
[user1@localhost ch5]$ PS1='LINUX ] '
LINUX ]
```

06 프롬프트 설정

■ 프롬프트를 다양한 형태로 바꿔보기

② 환경 변수를 사용. 다음은 프롬프트에 현재 작업 디렉터리가 출력되도록 한 것. 이 경우 디렉터리를 이동하면 프롬프트의 내용도 자동으로 바뀜

```
LINUX ] PS1='[$PWD] '
[/home/user1/Test/ch5]
```

③ 특수문자 ` `를 이용하여 명령의 실행 결과를 사용. uname -n 명령은 호스트 이름을 출력하는데 여기에 프롬프트의 끝을 표시하기 위해 $ 기호를 추가함

```
[/home/user1/Test/ch5] PS1='`uname -n` $ '
localhost.localdomain $
```

④ 이스케이프 문자 \u, \T, !를 사용

```
localhost.localdomain $ PS1='[\u \T] \!$ '
[user1 09:52:13] 483$
```

06 프롬프트 설정

■ 컬러 프롬프트 정의하기

컬러 프롬프트

- 형식 `PS1= '\[\e[x;y;nm\] 프롬프트 \[\e[x;y;0m\]'`

- 프롬프트의 첫 번째 문자 `\[`는 화면에 출력하지 않을 문자의 시작을 알려줌. 이렇게 하지 않으면 이스케이프 문자까지 모두 카운트되어 자동 줄 바꿈이 이상하게 동작함
- 프롬프트 앞에서 지정한 색상과 기능은 프롬프트의 색상과 특수 효과를 나타내며, 프롬프트 뒤에 지정한 색상은 원래 색상으로 돌아가기 위한 것. 이렇게 지정하지 않으면 이후 셸의 모든 출력이 프롬프트의 색상과 같아짐
- 특수 기능으로 역상을 이용할 때는 문자와 배경의 색상을 각각 지정해야 하며, 역상을 이용하지 않을 때는 한 가지 색상만 지정하면 됨

06 프롬프트 설정

■ 컬러 프롬프트 정의하기

표 5-13 프롬프트 컬러 번호

컬러	글자색 번호	배경색 번호
검은색	30	40
빨간색	31	41
녹색	32	42
갈색	33	43
파란색	34	44
보라색	35	45
청록색	36	46
흰색	37	47

표 5-14 프롬프트 특수 기능 번호

번호	기능
0	기본 색
1	굵게
4	흑백에서 밑줄
5	반짝임
7	역상
10	기본 폰트
38	밑줄 사용 가능
39	밑줄 사용 불가능

06 프롬프트 설정

■ 컬러 프롬프트 설정하기

① 파란색으로 설정

```
[user1 09:52:13] 483$ PS1='\e[34mLinux $\e[0;0m '
Linux $                ➞ 파란색
```

② 파란색 볼드로 설정

```
Linux $ PS1='\e[34;1mLinux $\e[0;0m '
Linux $                ➞ 파란색, 볼드
```

③ 밑줄 친 빨간색으로 설정

```
Linux $ PS1='\e[31;4mLinux $\e[0;0m '
Linux $                ➞ 빨간색, 밑줄
```

④ 배경은 갈색, 글자는 보라색, 프롬프트는 '사용자명@호스트명 $'로 설정

```
Linux $ PS1='\e[35;43m\u@\h \\e[0,0m '
user1@localhost $    ➞ 갈색 배경, 보라색 글자
```

06 프롬프트 설정

■ 출력하지 않을 문자열 설정 추가하기

- 컬러로 바꾼 프롬프트 상태에서 문자를 계속 입력 했을 때

```
aaaa1@localhost $ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

→ 중간에 이동하고 줄 바꿈도 안 한다.

- 터미널의 화면 끝에 가기 전에 커서가 앞으로 돌아오고 같은 행을 겹쳐서 기록. 언제 줄 바꿈을 해야 하는지 배시셸이 계산할 때, 프롬프트를 설정하면서 지정한 이스케이프 문자도 카운트하기 때문
- 프롬프트로 지정한 문자와 함께 사용자에게 보이지는 않지만 이스케이프 문자도 모두 카운트되어 터미널 창의 길이를 계산하는 데 반영된 것
- 이 문제를 해결하려면 \[와 \]를 사용하여 출력되지 않을 문자를 표시해야 함
- 셸이 터미널 창의 길이를 계산할 때 \[와 \] 사이의 글자는 제외하게 됨

06 프롬프트 설정

■ 출력하지 않을 문자열 설정 추가하기

- 프롬프트를 제대로 설정하기

```
user1@localhost $ PS1='\[\e[35;43m\]\u@\h $\[\e[0,0m\] '
```

- 정상적으로 줄 바꿈이 되는지 다시 문자를 입력해서 확인

```
user1@localhost $ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaa^C
user1@localhost $
```

- 프롬프트를 처음에 저장했던 값으로 되돌리기

```
user1@localhost $ PS1=$PROMPT
[user1@localhost ch5]$
```

07 환경 설정 파일

07 환경 설정 파일

■ 시스템 환경 설정 파일

- 시스템 환경 설정 파일은 시스템을 사용하는 전체 사용자의 공통 환경을 설정하는 파일로 모든 사용자가 로그인할 때마다 무조건 실행됨
- 시스템 환경 설정 파일은 일반 사용자가 수정할 수 없으며 시스템 관리자가 관리함

표 5-15 배시셸의 시스템 환경 설정 파일

파일	기능
/etc/profile	• 시스템에 공통으로 적용되는 환경 변수를 설정한다. – PATH: 기본 명령 경로를 설정한다. – USER, LOGNAME: 사용자 UID와 이름을 설정한다. – HOSTNAME: 호스트명을 설정한다. – HISTSIZE: 히스토리 크기를 설정한다. • /etc/profile.d/*.sh를 실행한다. • /etc/bashrc 파일을 실행한다.
/etc/bashrc	• 시스템 공통으로 적용되는 함수와 앨리어스를 설정한다. • 기본 프롬프트를 설정한다. • 서브 셸을 위한 명령 경로를 설정한다. • 서브 셸을 위한 기본 접근 권한을 설정한다.
/etc/profile.d/*.sh	• 언어나 명령별로 각각 필요한 환경을 설정한다. • 필요시 설정 파일을 추가한다.

07 환경 설정 파일

■ 시스템 환경 설정 파일

- /etc/profile 파일의 앞부분 일부를 출력해 보면 다음과 같음

```
[user1@localhost ch5]$ cd
[user1@localhost ~]$ more /etc/profile
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc
(생략)
pathmunge () {
    case ":${PATH}:" in
       *:"$1":*)
            ;;
(생략)
```

07 환경 설정 파일

■ 사용자 환경 설정 파일

- 사용자 환경 설정 파일은 각 사용자의 홈 디렉터리에 숨김 파일로 존재하며 사용자가 내용을 수정하고 관리할 수 있음
- 사용자가 로그인하면 제일 먼저 시스템 환경 설정 파일이 실행되어 시스템 공통 환경을 만들고, 이후 사용자 환경 설정 파일을 순서대로 실행하여 사용자별 환경을 설정함

표 5-16 배시셸의 사용자 환경 설정 파일

파일	기능
~/.bash_profile	• .bashrc 파일이 있으면 실행한다. • 경로 추가 등 사용자가 정의하는 환경 설정 파일이다.
~/.bashrc	• /etc/bashrc 파일이 있으면 실행한다. • 사용자가 정의하는 앨리어스나 함수 등을 설정한다.
~/.bash_logout	• 로그아웃 시 실행할 필요가 있는 함수 등을 설정한다.

07 환경 설정 파일

■ 사용자 환경 설정 파일 만들기

- vi로 .bashrc 파일을 열고 앞에서 실습한 앨리어스를 입력해 보기

```
[user1@localhost ~]$ vi .bashrc
# .bashrc
(생략)
# User specific aliases and functions
if [ -d ~/.bashrc.d ]; then
    for rc in ~/.bashrc.d/*; do
       if [ -f "$rc" ]; then
            . "$rc"
       fi
    done
fi

unset rc
alias rm='rm -i'     → 추가한 내용
alias ls='ls -F '    → 추가한 내용
~
:wq                  → 저장하고 종료한다.
```

07 환경 설정 파일

■ 사용자 환경 설정 파일 적용하기

- 사용자 환경 설정 파일을 수정한 다음 이를 적용하려면 로그아웃했다가 다시 로그인해야 함
- 다음과 같이 .(마침표) 명령이나 source 명령을 사용하여 환경 설정 파일을 실행해야 함

```
[user1@localhost ~]$ . .bashrc          → .(마침표) 명령 실행
```

또는

```
[user1@localhost ~]$ source .bashrc     → source 명령 실행
```

07 환경 설정 파일

■ 다른 셸의 환경 설정 파일

- 배시셸 외에 본셸, 콘셸, C셸 등도 각각 환경 설정 파일을 제공함

표 5-17 다른 셸의 환경 설정 파일

셸	시스템 초기화 파일	사용자 초기화 파일	실행 조건	실행 시기		
				로그인	서브 셸	로그아웃
본셸	/etc/profile	$HOME/.profile	–	○		
콘셸	/etc/profile	$HOME/.profile	–	○		
		$HOME/.kshrc	ENV 변수 설정	○	○	
C셸	/etc/.login	$HOME/.login	–	○		
		$HOME/.cshrc	–	○	○	
		$HOME/.logout	–			○

07 환경 설정 파일

■ 사용자 환경 설정 파일 수정하기

① .bash_profile 파일을 vi로 열기

```
[user1@localhost ~]$ vi .bash_profile
```

② .bash_profile 파일의 내용에 행 번호가 보이도록 함

```
  1 # .bash_profile
  2
  3 # Get the aliases and functions
  4 if [ -f ~/.bashrc ]; then
  5             . ~/.bashrc
  6 fi
  7
  8 # User specific environment and startup programs
~
:set nu
```

07 환경 설정 파일

■ 사용자 환경 설정 파일 수정하기

③ 8행 다음에 환경 변수 HISTSIZE를 추가하고 크기를 1500으로 설정한 뒤 파일을 저장한 후 종료

```
(생략)
  8 # User specific environment and startup programs
  9 HISTSIZE=1500
~
:wq
```

07 환경 설정 파일

■ 사용자 환경 설정 파일 수정하기

④ .bashrc 파일을 vi로 열고 행 번호가 보이도록 함

```
[user1@localhost ~]$ vi .bashrc
  1 # .bashrc
  2
  3 # Source global definitions
  4 if [ -f /etc/bashrc ]; then
  5             . /etc/bashrc
  6 fi
  7
  8 # User specific environment
  9 if ! [[ "$PATH" =~ "$HOME/.local/bin:$HOME/bin:" ]]
 10 then
 11    PATH="$HOME/.local/bin:$HOME/bin:$PATH"
 12 fi
 13 export PATH
(생략)
```

07 환경 설정 파일

■ 사용자 환경 설정 파일 수정하기

⑤ 12행 다음에 PATH 변수에 /etc 경로를 추가하고 파일을 저장한 후 종료. 경로를 추가할 때 구분자로 :(콜론)을 입력

```
(생략)
 12 fi
 13 PATH="$PATH:/etc"
 14 export PATH
(생략)
```

⑥ .bashrce 파일을 실행. 경로에 /etc 디렉터리가 추가되었는지 확인

```
[user1@localhost ~]$ source .bashrc
[user1@localhost ~]$ echo $PATH
/home/user1/.local/bin:/home/user1/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/
sbin:/etc
```

07 환경 설정 파일

■ 사용자 환경 설정 파일 수정하기

혼자해보기 사용자 환경 설정 파일 수정하기

❶ .bash_profile 파일을 vi로 연다.
❷ .bash_profile 파일에서 경로의 마지막에 추가한 HISTSIZE 변수를 삭제하고 파일을 저장한 후 종료한다.
❸ vi로 .bashrc 파일을 연다.
❹ .bashrc 파일에 다음 앨리어스를 추가한다.
  – clear 명령을 c로 앨리어스 생성
  – history 명령을 h로 앨리어스 생성
❺ [따라해보기]에서 추가한 13행을 삭제한다.
❻ 앨리어스 c와 h가 동작하는지 확인한다.
❼ 경로에서 /etc가 삭제되었는지 확인한다.

Thank you!

한빛아카데미 Hanbit Academy, Inc.