리눅스

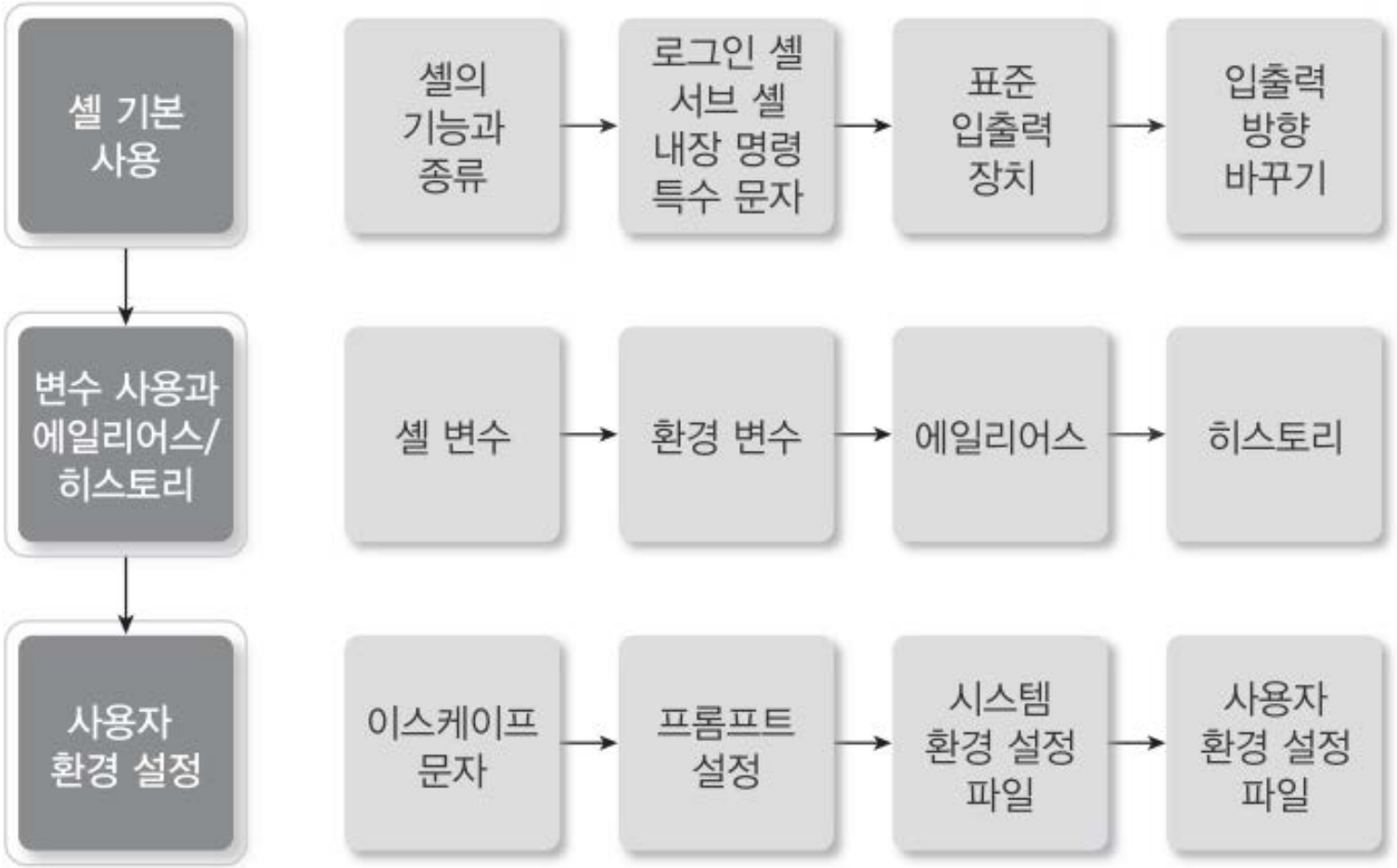
- 순　　서 : 제6장 (Shell)
- 학　　기 : 2019학년도 1학기
- 학　　과 : 가천대학교 컴퓨터공학과 2학년
- 교　　수 : 박 양 재

목차

학습목표

- 셸의 기능을 설명하고 주요 셸의 종류를 나열할 수 있다.
- 로그인 셸을 다른 셸로 바꿀 수 있다.
- 셸 특수 문자의 종류를 이해하고 필요에 따라 적절하게 사용할 수 있다.
- 표준 입출력 장치를 이해하고 입출력 방향 바꾸기를 할 수 있다.
- 셸 변수와 환경 변수의 차이를 이해하고 변수를 정의하여 사용할 수 있다.
- 새로운 에일리어스를 만들거나 필요 없는 에일리어스를 해제할 수 있다.
- 히스토리 기능으로 명령을 재실행할 수 있다.
- 이스케이프 문자를 이해하고 프롬프트를 원하는 형태로 바꿀 수 있다.
- 시스템 환경 설정 파일과 사용자 환경 설정 파일을 구분하고 사용자 환경을 설정할 수 있다.

00 개요


셸 기본 사용
셸의 기능과 종류
로그인 셸 서브 셸 내장 명령 특수 문자
표준 입출력 장치
입출력 방향 바꾸기
변수 사용과 에일리어스/ 히스토리
셸 변수
환경 변수
에일리어스
히스토리
사용자 환경 설정
이스케이프 문자
프롬프트 설정
시스템 환경 설정 파일
사용자 환경 설정 파일

01 셸의 기능과 종류

- 셸의 기능
  - 명령어 해석기 기능, 프로그래밍 기능, 사용자 환경 설정 기능

- 명령어 해석기 기능
  - 사용자와 커널 사이에서 명령을 해석하여 전달하는 해석기(**interpreter**)와 번역기(**translator**) 기능
  - 사용자가 로그인하면 셸이 자동으로 실행되어 사용자가 명령을 입력하기를 기다림 -> 로그인 셸
  - 로그인 셸은 **/etc/passwd** 파일에 사용자 별로 지정
  - 프롬프트: 셸이 사용자의 명령을 기다리고 있음을 나타내는 표시
- 프로그래밍 기능
  - 셸은 자체 내에 프로그래밍 기능이 있어 반복적으로 수행하는 작업을 하나의 프로그램으로 작성 가능
  - 셸 프로그램을 셸 스크립트
- 사용자 환경 설정 기능
  - 사용자 환경을 설정할 수 있도록 초기화 파일 기능을 제공
  - 초기화 파일에는 명령을 찾아오는 경로를 설정하거나, 파일과 디렉터리를 새로 생성할 때 기본 권한을 설정하거나, 다양한 환경 변수 등을 설정

01 셸의 기능과 종류

- 셸의 종류
  - 본 셸, 콘 셸, C 셸, 배시 셸

- 본 셸(Bourne shell)
  - 유닉스 V7에 처음 등장한 최초의 셸
  - 개발자의 이름인 스티븐 본(Stephen Bourne)의 이름을 따서 본 셸이라고 함
  - 본 셸의 명령 이름은 sh임
  - 초기에 본 셸은 단순하고 처리 속도가 빨라서 많이 사용되었고, 지금도 시스템 관리 작업을 수행하는 많은 셸 스크립트는 본 셸을 기반으로 하고 있음
  - 히스토리, 에일리어스, 작업 제어 등 사용자의 편의를 위한 기능을 제공하지 못해 이후에 다른 셸들이 등장
  - 페도라 에서 본 셸의 경로를 확인해보면 배시 셸과 심볼릭 링크로 연결되어 있음

```
[park@localhost ~]$
[park@localhost ~]$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 2013-10-10 01:09 /bin/sh -> bash
[park@localhost ~]$
```

01 셸의 기능과 종류

- C 셸(C shell)
  - 캘리포니아대학교(버클리)에서 빌 조이(Bill Joy)가 개발
  - 2BSD 유닉스에 포함되어 발표
  - 본 셸에는 없던 에일리어스나 히스토리 같은 사용자 편의 기능을 포함
  - 셸 스크립트 작성을 위한 구문 형식이 C 언어와 같아 C 셸이라는 이름을 가지게 되었음
  - C 셸의 명령 이름은 csh
- 콘 셸(Korn shell)
  - 1980년대 중반 AT&T 벨연구소의 데이비드 콘(David Korn)이 콘 셸을 개발
  - 유닉스 SVR 4에 포함되어 발표
  - C 셸과 달리 본 셸과의 호환성을 유지하고 히스토리, 에일리어스 기능 등 C 셸의 특징도 모두 제공하면서 처리 속도도 빠름
  - 콘셸의 명령 이름은 ksh
- 배시 셸(bash shell)
  - 본 셸을 기반으로 개발된 셸로서 1988년 브레인 폭스(Brain Fox)가 개발
  - 본 셸과 호환성을 유지하면서 C 셸, 콘 셸의 편리한 기능도 포함
  - 배시 셸의 명령 이름은 bash
  - 배시 셸의 모든 버전은 GPL 라이선스에 의거하여 자유롭게 사용 가능 리눅스의 기본 셸

02 셸 기본 사용법

- 기본 셸 확인
  - 프롬프트 모양 참조
    - 본 셸, 배시 셸, 콘 셸의 기본 프롬프트: **$**
    - **C** 셸의 기본 프롬프트: **%**
  - 사용자 정보 확인: **/etc/passwd** 파일
    - 사용자 정보의 가장 마지막에 나온 **/bin/bash**가 기본 셸

```
[user1@localhost ~]$ grep user1 /etc/passwd
user1:x:1001:1001::/home/user1:/bin/bash
[user1@localhost ~]$
```

02 셸 기본 사용법

기본 셸 바꾸기

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

- 바꿀 수 있는 셸의 종류: **/etc/shells** 파일에 지정

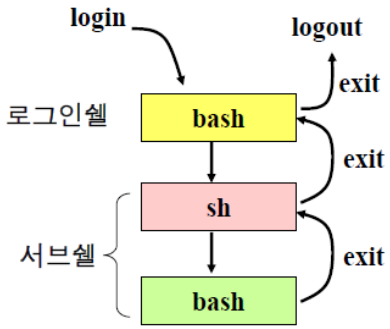
```
[user1@localhost ~]$
[user1@localhost ~]$ cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
/bin/tcsh
/bin/csh
[user1@localhost ~]$ _
```

02 셸 기본 사용법

기본 셸 바꾸기 예

- 바꾸려는 셸은 절대 경로로 지정해야 함.
- 현재 사용중인 셸 확인

```
[user1@gachon ~]$ echo $SHELL
/bin/bash
```




```
[user1@localhost ~]$ cat /etc/passwd | grep user1
user1:x:1001:1001::/home/user1:/bin/bash
[user1@localhost ~]$
[user1@localhost ~]$
[user1@localhost ~]$ chsh -s /bin/sh user1
Changing shell for user1.
암호:
Shell changed.
[user1@localhost ~]$
[user1@localhost ~]$
[user1@localhost ~]$ cat /etc/passwd | grep user1
user1:x:1001:1001::/home/user1:/bin/sh
[user1@localhost ~]$
[user1@localhost ~]$
[user1@localhost ~]$ chsh -s /bin/bash user1
Changing shell for user1.
암호:
Shell changed.
[user1@localhost ~]$ cat /etc/passwd | grep user1
user1:x:1001:1001::/home/user1:/bin/bash
[user1@localhost ~]$
```

로그인 셸과 서브 셸

- 로그인셸 : 사용자가 로그인한 직후 자동 생성되는 셸
- 서브셸 : 사용자가 직접 실행한 셸
- 서브 셸을 종료하는 명령은 **^d(ctrl +d), exit** 등 사용
- 서브 셸이 종료되면 서브 셸을 실행했던 이전 셸 환경으로 복귀
- 로그인 셸에서 로그아웃하면 접속 해제

02 셸 기본 사용법

- 기본 셸 바꾸기 예

```
[user1@localhost ~]$ chsh -s /bin/csh user1
Changing shell for user1.
암호:
Shell changed.
[user1@localhost ~]$
[user1@localhost ~]$ echo $SHELL
/bin/bash
[user1@localhost ~]$ _
```

- 로그아웃 후 셸 확인(변경된 셸)

```
[user1@localhost ~]$ echo $SHELL
/bin/csh
[user1@localhost ~]$
```

- 원래 리눅스 로그인셸(bash)로 변경 실습 -> 로그아웃 후 확인

```
[user1@localhost ~]$ chsh -s /bin/bash user1
Changing shell for user1.
암호:
Shell changed.
[user1@localhost ~]$
```

02 셸 기본 사용법

셸 내장 명령

- 셸은 자체적으로 내장 명령을 가지고 있음
- 셸 내장 명령은 별도의 실행 파일이 없고 셸 안에 포함
- 셸 명령 예:**/usr/bin/ yum**

```
[park@localhost ~]$ file /usr/bin/yum
/usr/bin/yum: python script text executable
[park@localhost ~]$
```

```
[park@localhost ~]$ cat /usr/bin/yum
#!/usr/bin/python
import sys
try:
    import yum
except ImportError:
    print >> sys.stderr, """\
There was a problem importing one of the Python modules
required to run yum. The error leading to this problem was:

   %s

Please install a package which provides this module, or
verify that the module is installed correctly.
```

- 일반 명령(실행 파일)의 경우 예 : **/usr/bin/bc**
  - 실행 파일은 바이너리 파일이므로 **cat** 명령으로 파일의 내용을 확인할 수 없음

```
[park@localhost ~]$ file /usr/bin/bc
/usr/bin/bc: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamical
ly linked (uses shared libs), for GNU/Linux 2.6.18, stripped
[park@localhost ~]$
```

02 셸 기본 사용법

- 배시 셸의 출력 명령
  - **echo**

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

```
[user1@localhost ~]$ echo linux
linux
[user1@localhost ~]$ echo "linux fedora"
linux fedora
[user1@localhost ~]$
```

02 셸 기본 사용법

배시 셸의 출력 명령

- **printf**
  - % 지시자와 \ 문자를 이용하여 출력 형식을 지정 가능

printf

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

```
[user1@localhost ~]$ printf linux
linux[user1@localhost ~]$ printf "linux fedora₩n"
linux fedora
[user1@localhost ~]$ printf "%d + %d = %d₩n" 10 10 20
10 + 10 = 20
[user1@localhost ~]$
```

02 셸 기본 사용법

특수 문자 사용하기

- 사용자가 더욱 편리하게 명령을 입력하고 실행할 수 있도록 다양한 특수 문자를 제공
- 주요 특수 문자는 *, ?, |, ;, [ ], ~, ‘ ’, “ ”, ` `` 등
- 명령을 입력하면 셸은 먼저 특수 문자가 있는지 확인하고 이를 적절한 형태로 변경한 후 명령을 실행

특수 문자 *(별표)

- 임의의 문자열을 나타내는 특수 문자로 0개 이상의 문자로 대체

특수 문자 *

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 시작하는 모든 파일의 이름과 파일 종류를 출력한다. t도 해당한다는 데 주의한다.
cp *.txt ../ch3	확장자가 txt인 모든 파일을 상위 디렉터리 밑의 ch3 디렉터리로 복사한다.
ls –l h*d	파일명이 h로 시작하고 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

- 특수 문자 ?와 [ ]
  - 하나의 문자를 나타내는 데 사용
  - ?는 길이가 1인 임의의 한 문자를, [ ]는 괄호 안에 포함된 문자 중 하나를 나타냄

특수 문자 ?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 제외된다.
ls –l tmp[135].txt	tmp 다음에 1, 3, 5 중 한 글자가 오고 파일의 확장자가 txt인 모든 파일의 이름을 출력한다. tmp1.txt, tmp3.txt, tmp5.txt 파일이 있으면 해당 파일의 상세 정보를 출력한다. tmp.txt는 제외된다.
ls –l tmp[1–3].txt	[1–3]은 1부터 3까지의 범위를 의미한다. 따라서 ls –l tmp[123].txt와 결과가 같다. tmp1.txt, tmp2.txt, tmp3.txt 파일이 있으면 해당 파일의 상세 정보를 출력한다.
ls [0–9]*	파일명이 숫자로 시작하는 모든 파일 목록을 출력한다.
ls [A–Za–z]*[0–9]	파일명이 영문자로 시작하고 숫자로 끝나는 모든 파일 목록을 출력한다.

02 셸 기본 사용법

- 특수 문자 ~와 –
  - ~(물결표)와 -(붙임표)는 디렉터리를 나타내는 특수 문자
  - ~만 사용하면 현재 작업 중인 사용자의 홈 디렉터리를 표시하고 다른 사용자의 로그인 ID와 함께 사용하면(~로그인 ID) 해당 사용자의 홈 디렉터리 표시
  - -는 cd 명령으로 디렉터리를 이전하기 직전의 작업 디렉터리 표시

특수 문자 ~와 –

사용 예	의미
cp *.txt ~/ch3	확장자가 txt인 모든 파일을 현재 작업 중인 사용자의 홈 디렉터리 아래 tmp 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

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

특수 문자 ;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

- 특수 문자 ‘ ’와 “ ”
  - ‘ ’(작은따옴표)와 “ ”(큰따옴표)는 문자를 감싸서 문자열로 만들어주고, 문자열 안에 사용된 특수 문자의 기능을 없앰
  - ‘ ’는 모든 특수 문자를, “ ”는 $, ``, \을 제외한 모든 특수 문자를 일반 문자로 간주하여 처리

특수 문자 ‘ ’와 “ ”

사용 예	의미
echo ‘$SHELL’	$SHELL 문자열이 화면에 출력된다.
echo “$SHELL”	셸 환경 변수인 SHELL에 저장된 값인 현재 셸의 종류가 화면에 출력된다. 예를 들면 /bin/sh이다.

- 특수 문자 ` `(백쿼테이션-1자 좌측 키)
  - 셸은 `` ``로 감싸인 문자열을 명령으로 해석하여 명령의 실행 결과로 전환

특수 문자 ` `

사용 예	의미
echo “Today is `date`”	`date` 는 명령으로 해석되어 date 명령의 실행 결과로 바뀐다. 결과적으로 다음과 같이 화면에 출력된다. Today is 2013. 03. 24. (일) 02:37:52 KST
ls /platform/`uname –m`	uname –m 명령의 실행 결과를 문자열로 바꿔 디렉터리 이름으로 사용한다.

02 셸 기본 사용법

특수 문자 ₩

- \(역빗금, \와 동일함)은 특수 문자 바로 앞에 사용되는데 해당 특수 문자의 효과를 없애고 일반 문자처럼 처리

특수 문자 \

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'의 결과와 같다.

특수 문자 >, <, >>

- 입출력의 방향을 바꾸는 특수 문자

특수 문자 〉, 〈, 〉〉

사용 예	의미
ls -l 〉res	ls -l 명령의 실행 결과를 화면이 아닌 res 파일에 저장한다.

03 입출력 방향 바꾸기

표준 입출력 장치

- 표준 입력 장치: 셸이 작업을 수행하는 데 필요한 정보를 받아들이는 장치 -> 키보드
- 표준 출력 장치: 실행 결과를 내보내는 장치 -> 모니터
- 표준 오류 장치: 오류 메시지를 내보내는 장치 -> 모니터

파일 디스크립터

- 파일 관리를 위해 붙이는 일련 번호
- 입출력 장치를 변경할 때는 이 파일 디스크립터를 사용
- 표준 입출력 장치를 파일로 바꾸는 것을 '리다이렉션(redirection)'이라고 함

표준 입출력 장치의 파일 디스크립터

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

03 입출력 방향 바꾸기

- 출력 리다이렉션
  - > : 기존 파일의 내용을 삭제하고 새로 결과를 저장
  - >> : 기존 파일의 내용 뒤에 결과를 추가
- 파일 덮어쓰기 : >

>	
기능	파일 리다이렉션(덮어쓰기)
형식	명령 1> 파일 이름 명령 > 파일 이름

  - 1: 파일 디스크립터 1번(표준 출력, 화면)
  - 셸은 >를 사용한 리다이렉션에서 지정한 이름의 파일이 없으면 파일을 생성해서 명령의 수행 결과를 저장
  - 파일이 있으면 이전의 내용이 없어지고 명령의 수행 결과로 대체

03 입출력 방향 바꾸기

- 파일 덮어쓰기 : > 예

```
[user1@localhost ~]$ mkdir -p linux/ch6
[user1@localhost ~]$
[user1@localhost ~]$
[user1@localhost ~]$ ls
linux     ts1111   tww   다운로드   바탕화면   사진   음악
services  tslslsl  공개  문서       비디오     서식
[user1@localhost ~]$
[user1@localhost ~]$ cd linux/ch6/
[user1@localhost ch6]$ ls -la
합계 0
drwxrwxr-x 2 user1 user1  6  4월   4 10:48 .
drwxrwxr-x 3 user1 user1 17  4월   4 10:48 ..
[user1@localhost ch6]$ ls -la > out1
[user1@localhost ch6]$
[user1@localhost ch6]$ cat out1
합계 0
drwxrwxr-x 2 user1 user1 18  4월   4 10:49 .
drwxrwxr-x 3 user1 user1 17  4월   4 10:48 ..
-rw-rw-r-- 1 user1 user1  0  4월   4 10:49 out1
```

03 입출력 방향 바꾸기

- 예상치 않게 파일의 내용이 겹쳐 쓰이는 상황을 예방하기

```
[user1@localhost ch6]$ set -o noclobber
[user1@localhost ch6]$ ls > out1
-bash: out1: cannot overwrite existing file
[user1@localhost ch6]$ _
```

  - 설정 해제

```
[user1@localhost ch6]$ set +o noclobber
[user1@localhost ch6]$ ls > out1
[user1@localhost ch6]$
```

- cat 명령으로 파일 생성하기

```
[user1@localhost ch6]$ cat > out1
Linux CentOS
I love Linux

[user1@localhost ch6]$ cat out1
Linux CentOS
I love Linux

[user1@localhost ch6]$ _
```

03 입출력 방향 바꾸기

- 파일에 내용 추가하기 : >>

>>	
기능	파일에 내용을 추가한다.
형식	명령 >> 파일 이름

  - 지정한 파일이 없으면 파일을 생성하고, 파일이 있으면 기존 파일의 끝에 명령의 실행 결과를 추가

```
[user1@localhost ch6]$ cat out1
Linux CentOS
I love Linux

[user1@localhost ch6]$
[user1@localhost ch6]$ date >> out1
[user1@localhost ch6]$
[user1@localhost ch6]$ cat out1
Linux CentOS
I love Linux

2019. 04. 04. (목) 10:58:09 KST
[user1@localhost ch6]$
```

03 입출력 방향 바꾸기

- 오류 리다이렉션
  - 표준 오류도 기본적으로 화면으로 출력되며 표준 출력처럼 리다이렉션 가능
  - 표준 출력과 표준 오류 예

```
[user1@localhost ch6]$ ls
out1
[user1@localhost ch6]$
[user1@localhost ch6]$ ls /abc
ls: cannot access /abc: 그런 파일이나 디렉터리가 없습니다
[user1@localhost ch6]$
```

  - 표준출력 리다이렉션: 오류 메시지는 리다이렉션 안됨

```
[user1@localhost ch6]$ ls > ls.out
[user1@localhost ch6]$ ls /abc > ls.err
ls: cannot access /abc: 그런 파일이나 디렉터리가 없습니다
[user1@localhost ch6]$ cat ls.err
[user1@localhost ch6]$ cat ls.out
ls.out
out1
[user1@localhost ch6]$
```

표준 출력 리다이렉트

표준 출력 리다이렉트

오류 메세지 화면 출력

오류 메세지 저장 안됨

표준 출력 내용 출력

03 입출력 방향 바꾸기

오류 리다이렉션

- 오류 리다이렉션에서는 파일 디스크립터 번호를 생략 불가

2>	
기능	표준 오류 메시지를 파일에 저장한다.
형식	명령 2> 파일 이름

```
[user1@localhost ch6]$ ls /abc 2> ls.err
[user1@localhost ch6]$
[user1@localhost ch6]$ cat ls.err
ls: cannot access /abc: 그런 파일이나 디렉터리가 없습니다
[user1@localhost ch6]$
```

표준 오류를 리다이렉트

파일에 저장된 내용

- 표준 출력과 표준 오류를 한 번에 리다이렉션하기

```
[user1@localhost ch6]$ ls . /abc > ls.out 2> ls.err
[user1@localhost ch6]$
[user1@localhost ch6]$ cat ls.out
.:
ls.err
ls.out
out1
[user1@localhost ch6]$ cat ls.err
ls: cannot access /abc: 그런 파일이나 디렉터리가 없습니다
[user1@localhost ch6]$
```

03 입출력 방향 바꾸기

- 오류 리다이렉션
  - 오류 메시지 버리기

```
[user1@localhost ch6]$ ls /abc 2> /dev/null
[user1@localhost ch6]$
```

- 표준 출력과 표준 오류를 한 파일로 리다이렉션하기
  - 명령의 정상 실행 결과를 파일로 리다이렉션(>).
  - 그 명령 전체의 오류 메시지를 1번 파일(표준 출력 파일, &1이라고 표현함)로 리다이렉션(2>).

```
[user1@localhost ch6]$ ls . /abc > ls.out 2>&1
[user1@localhost ch6]$
[user1@localhost ch6]$
[user1@localhost ch6]$ cat ls.out
ls: cannot access /abc: 그런 파일이나 디렉터리가 없습니다
.:
ls.err
ls.out
out1
[user1@localhost ch6]$
```

오류 메시지 저장

현재 지렉토리 내용

03 입출력 방향 바꾸기

입력 리다이렉션

<	
기능	표준 입력을 바꾼다.
형식	명령 0< 파일 이름 명령 < 파일 이름

- 입력 리다이렉션 예: **cat** 명령

```
[user1@localhost ch6]$ cat out1
Linux CentOS
I love Linux

2019. 04. 04. (목) 10:58:09 KST
[user1@localhost ch6]$
[user1@localhost ch6]$ cat < out1
Linux CentOS
I love Linux

2019. 04. 04. (목) 10:58:09 KST
[user1@localhost ch6]$
[user1@localhost ch6]$ cat 0< out1
Linux CentOS
I love Linux

2019. 04. 04. (목) 10:58:09 KST
[user1@localhost ch6]$
```

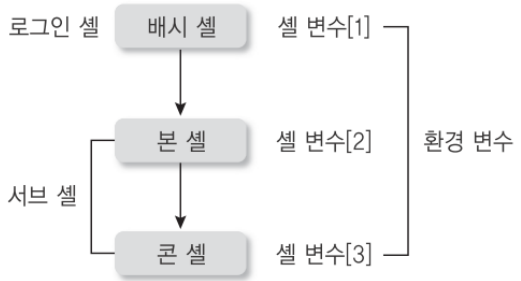
파일내용 출력(< 생략)

표준 입력을 리디렉션

표준 입력을 리디렉션내용

04 배시 셸 환경 설정

- 셸 변수와 환경변수
  - 셸의 환경을 설정하기 위한 값을 저장할 수 있도록 셸 변수와 환경 변수를 제공
  - 셸 변수: 현재 셸에서만 사용이 가능하고 서브 셸로는 전달되지 않음(지역변수)
  - 환경 변수: 현재 셸뿐만 아니라 서브 셸로도 전달(전역변수)



[그림 4-4] 셸 변수와 환경 변수

- 전체 변수 출력: set, env
  - **set:** 셸 변수와 환경변수 모두 출력

```
[user1@localhost ch6]$
[user1@localhost ch6]$ set
ABRT_DEBUG_LOG=/dev/null
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extglob:extquote:force_fignore:hist
append:interactive_comments:login_shell:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_COMPAT_DIR=/etc/bash_completion.d
BASH_LINENO=()
BASH_REMATCH=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="2" [2]="46" [3]="2" [4]="release" [5]="x86_64-redhat
-linux-gnu")
BASH_VERSION='4.2.46(2)-release'
COLUMNS=80
COMP_WORDBREAKS=$' ₩t₩n"₩><=;|&(:'
DIRSTACK=()
EUID=1001
```

04 배시 셸 환경 설정

- 전체 변수 출력: set, env
  - **env:** 환경변수만 출력

```
[user1@localhost ch6]$ env
XDG_VTNR=1
XDG_SESSION_ID=1
HOSTNAME=localhost.localdomain
SHELL=/bin/bash
TERM=xterm-256color
HISTSIZE=1000
NLS_LANG=KOREAN_KOREA.AL32UTF8
QTDIR=/usr/lib64/qt-3.3
QTINC=/usr/lib64/qt-3.3/include
QT_GRAPHICSSYSTEM_CHECKED=1
USER=user1
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:pi=40;38;5;11:so=38;5;13:do=3
8;5;5:bd=48;5;232;38;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=05;48;5;232;3
8;5;15:su=48;5;196;38;5;15:sg=48;5;11;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38
;5;16:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;34:*.tar=38;5;9:*.tgz=38;5;9
:*.arc=38;5;9:*.arj=38;5;9:*.taz=38;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.lzh=38;5;9:*
.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5;9:*.tzo=38;5;9:*.t7z=38;5;9:*.zip=38;5;9:*.
z=38;5;9:*.Z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38;5;9:*.lz=38;5;9:*.lzo=38;5;
9:*.xz=38;5;9:*.bz2=38;5;9:*.bz=38;5;9:*.tbz=38;5;9:*.tbz2=38;5;9:*.tz=38;5;9:*.
```

04 배시 셸 환경 설정

- 주요 셸 환경변수

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 디렉터리 절대 경로
LANG	사용하는 언어	SHELL	로그인 셸
LOGNAME	사용자 계정 이름		

- 특정 변수 출력하기 : echo
  - 변수의 값을 출력할 때는 변수 이름 앞에 특수 문자 $를 붙임

```
[user1@localhost ch6]$ echo $SHELL
/bin/bash
[user1@localhost ch6]$
```

04 배시 셸 환경 설정

- 셸 변수 설정하기
  - 변수 이름과 문자열 사이에 공백이 있으면 안됨

셸 변수 설정	
형식	변수 이름=문자열
사용 예	SOME=test

```
[user1@localhost ch6]$ SOME=test
[user1@localhost ch6]$ echo $SOME
test
[user1@localhost ch6]$
```

04 배시 셸 환경 설정

- 환경 변수 설정하기 : export
  - 먼저 셸 변수를 정의하고, **export** 명령을 사용하여 이를 환경 변수로 변경

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

```
[user1@localhost ch6]$ export SOME
[user1@localhost ch6]$ env
XDG_VTNR=1
XDG_SESSION_ID=1
HOSTNAME=localhost.localdomain
SHELL=/bin/bash
TERM=xterm-256color
HISTSIZE=1000
SOME=test
NLS_LANG=KOREAN_KOREA.AL32UTF8
```

  - 변수를 설정하면서 바로 **export** 명령을 사용하여 한 번에 환경 변수로 전환도 가능

```
[user1@localhost ch6]$ export SOME1=test1
[user1@localhost ch6]$ echo $SOME1
test1
[user1@localhost ch6]$
```

04 배시 셸 환경 설정

- 환경 변수를 다시 셸 변수로 바꾸기 : export –n
  - 예: SOME은 보이지만 SOME1은 보이지 않음

```
[user1@localhost ch6]$ export -n SOME1
[user1@localhost ch6]$ env
XDG_VTNR=1
XDG_SESSION_ID=1
HOSTNAME=localhost.localdomain
SHELL=/bin/bash
TERM=xterm-256color
HISTSIZE=1000
SOME=test
NLS_LANG=KOREAN_KOREA.AL32UTF8
```

- 변수 해제하기

unset	
기능	지정한 변수를 해제한다.
형식	unset 변수
사용 예	unset SOME

```
[user1@localhost ch6]$ export SOME1=test1
[user1@localhost ch6]$
[user1@localhost ch6]$ echo $SOME1
test1
[user1@localhost ch6]$ unset SOME1
[user1@localhost ch6]$
[user1@localhost ch6]$ echo $SOME1

[user1@localhost ch6]$ _
```

05 에일리어스와 히스토리

에일리어스

- 에일리어스(alias)는 우리말로 '별명'을 의미
- 기존의 명령을 대신하여 다른 이름(별명)을 붙일 수 있도록 하는 기능
- 긴 명령 대신 짧은 명령을 만들어 사용 가능
- 여러 명령을 연결하여 하나의 명령으로 만들 수도 있음
- 자주 사용하는 옵션을 포함하여 새로운 이름을 붙여서 사용 가능

alias

기능	에일리어스를 생성한다.
형식	alias 이름='명령'
사용 예	alias : 현재 설정된 별칭 목록 출력 alias 이름='명령' : 명령을 수정하여 사용하는 경우 alias 이름='명령;명령2;...' : 여러 명령을 하나의 이름으로 사용하는 경우

05 에일리어스와 히스토리

- 기존 에일리어스 확인: alias
  - 아무것도 지정하지 않고 **alias** 명령을 실행하면 현재 설정되어 있는 에일리어스가 출력

```
[user1@localhost ch6]$ 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'
alias ls='ls --color=auto'
alias mc='. /usr/libexec/mc/mc-wrapper.sh'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-ti
lde'
[user1@localhost ch6]$
```

  - 에일리어스 실행 예

```
[user1@localhost ch6]$ ls
ls.err  ls.out  out1
[user1@localhost ch6]$ l.
.  ..
[user1@localhost ch6]$ ll
합계 12
-rw-rw-r-- 1 user1 user1 73  4월  4 11:10 ls.err
-rw-rw-r-- 1 user1 user1 95  4월  4 11:16 ls.out
-rw-rw-r-- 1 user1 user1 60  4월  4 10:58 out1
[user1@localhost ch6]$
```

05 에일리어스와 히스토리

- 에일리어스 설정하기 : alias
  - '에일리어스 이름=명령' 형식 사용
  - 에일리어스 설정 예: **ls**

```
[user1@localhost ch6]$ mkdir tmp
[user1@localhost ch6]$
[user1@localhost ch6]$ ls
ls.err  ls.out  out1  tmp
[user1@localhost ch6]$
[user1@localhost ch6]$ alias ls='ls -F'
[user1@localhost ch6]$
[user1@localhost ch6]$ ls
ls.err  ls.out  out1  tmp/
[user1@localhost ch6]$
```

  - 에일리어스 설정 예: **rm**

```
[user1@localhost ch6]$ alias rm='rm -i'
[user1@localhost ch6]$ rm out1
rm: remove 일반 파일 `out1'? n
[user1@localhost ch6]$
```

05 에일리어스와 히스토리

- 에일리어스에 인자 전달하기
  - 배시 셸에서는 에일리어스로 인자를 전달할 수 없음
  - 배시 셸에서 인자를 전달하려면 프로그래밍 기능에서 함수를 사용
  - 인자 전달 함수 예 **(cd $1 : 첫번째 인자 값)**

```
[user1@localhost ch6]$ function cdpwd {
> cd $1; pwd
> }
[user1@localhost ch6]$ cdpwd /tmp
/tmp
[user1@localhost tmp]$
```

- 에일리어스 해제하기 : unalias

unalias	
기능	에일리어스를 삭제한다.
형식	unalias 에일리어스

```
[user1@localhost ~]$ unalias ls
[user1@localhost ~]$ 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'
```

```
[user1@localhost ~]$ alias ls='ls -l --color=auto'
[user1@localhost ~]$ 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'
alias ls='ls -l --color=auto'
```

05 에일리어스와 히스토리

- 히스토리
  - 사용자가 이전에 입력한 명령을 다시 불러 사용하는 것

history

기능 히스토리(명령 입력 기록)를 출력한다.

형식 history

```
[user1@localhost ~]$ history
    1  ls -la
    2  ls
    3  exit
    4  ls -la
    5  ls
    6  ls -l
    7  ls -a
    8  exit
    9  ls
   10  ls -la
   11  ls -l
   12  ls -a
   13  ls
```

05 에일리어스와 히스토리

- 명령 재실행하기 : !

!를 사용한 명령 재실행 방법

재실행 방법	기능
!!	바로 직전에 수행한 명령을 재실행한다.
!번호	히스토리에서 해당 번호의 명령을 재실행한다.
!문자열	히스토리에서 해당 문자열로 시작하는 마지막 명령을 재실행한다.

- 직전 명령 재실행 예

```
[user1@localhost ~]$ cd ~/linux/ch6/
[user1@localhost ch6]$
[user1@localhost ch6]$ ls
ls.err  ls.out  out1  tmp
[user1@localhost ch6]$
[user1@localhost ch6]$ !!     직전 명령어 재실행
ls
ls.err  ls.out  out1  tmp
[user1@localhost ch6]$
```

05 에일리어스와 히스토리

- 명령 재실행하기 : !
  - 이전에 수행한 명령을 재실행 예

```
[user1@localhost ch6]$ history
    1  ls -la
    2  ls
    3  exit
    4  ls -la
    5  ls
    6  ls -l
    7  ls -a
    8  exit
    9  ls
   10  ls -la
   11  ls -l
   12  ls -a
   13  ls
   14  ls -la
   15  init 0
   16  exit
   17  su root
   18  ls
   19  ls -l
   20  ls -a
   21  ls -la
   22  ls -a /tmp
```

```
[user1@localhost ch6]$ !22
ls -a /tmp
.
..
.ICE-unix
.Test-unix
.X0-lock
.X11-unix
.XIM-unix
.esd-0
.esd-1001
.font-unix
```

! 사건번호 명령 재실행

```
[user1@localhost ch6]$ !l
ls -a /tmp
.
..
.ICE-unix
.Test-unix
.X0-lock
.X11-unix
.XIM-unix
.esd-0
.esd-1001
```

명령의 앞 글자로 재실행

05 에일리어스와 히스토리

명령 편집하기와 재실행하기

- 화살표 키를 사용하여 오류가 난 명령을 다시 프롬프트로 불러내서 수정한 뒤 재실행 가능

① 편집과 재실행 예1 : 명령에 오타를 입력

```
[park@localhost ch6]$ man hisdory
No manual entry for hisdory
[park@localhost ch6]$
```

② 프롬프트에서 ↑키를 누르면 방금 실행한 명령이 다시 나타남

```
[park@localhost ch6]$ man hisdory
```

③ 좌우 화살표로 커서를 이동하여 백스페이스키로 삭제한 후 다시 글자를 입력하고 엔터키를 눌러서 실행

```
[park@localhost ch6]$ man history
```

히스토리 저장하기

- 로그아웃할 때 홈 디렉터리 아래의 숨김 파일인 **.bash_history**에 히스토리 저장

```
[park@localhost ch6]$ more ~/.bash_history
gedit
ls
ls -a
ls /tmp
(생략)
[park@localhost ch6]$
```

06 프롬프트 설정

- 프롬프트 설정 변수: PS1
  - 프롬프트를 바꾸는 것은 환경 변수 **PS1**에 새로운 형태의 문자열을 지정하는 것

```
user1@localhost ch6]$ echo $PS1
₩u@₩h ₩W]₩$
user1@localhost ch6]$
```

PS1 변수에 현재 설정된 값

- 이스케이프 문자와 프롬프트 설정하기
  - \으로 시작하는 특별한 문자가 이스케이프 문자
  - **\u**와 같이 \으로 시작하는 이스케이프 문자는 두 글자가 아니라 한 글자로 처리
  - 이스케이프 문자는 화면에 문자 그대로 출력되지 않고 셸이 문자의 의미를 해석하여 실행

06 프롬프트 설정

- 프롬프트에서 사용할 수 있는 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 시각을 표시한다(HH:MM:SS 형식).
\T	12시간 형식으로 현재 시각을 표시한다(HH:MM:SS 형식).
\@	12시간 형식으로 현재 시각을 표시한다(오전/오후 형식).
\u	사용자 이름
\v	배시 셸의 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 끝 부분을 표시한다.

06 프롬프트 설정

프롬프트 변경 예제

① 간단한 문자열로 변경: 프롬프트의 끝을 표시하기 위해 마지막에]나 $ 같은 표시를 하고 공백 문자를 둠

```
[park@localhost ch6]$ PS1='LINUX ] '
LINUX ]
```

② 환경 변수를 사용: 프롬프트에 현재 작업 디렉터리가 출력

```
LINUX ] PS1='[$PWD] '
[/home/park/linux/ch6] cd ..
[/home/park/linux]
```

③ 명령의 실행 결과를 사용: 특수 문자 ``를 이용, **uname -n** 명령은 호스트 이름을 출력

```
[/home/park/linux]  PS1='`uname -n` #! $ '
localhost.localdomain 852$
```

④ 이스케이프 문자 **\u, \T, \!**를 사용

```
localhost.localdomain $ PS1='[\u \T] \!$ '
[park 09:14:36] 854$
```

06 프롬프트 설정

- 컬러 프롬프트 설정하기

컬러 프롬프트

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

프롬프트 컬러 번호

컬러	글자 색 번호	배경 색 번호	컬러	글자 색 번호	배경 색 번호
검은색	30	40	파란색	34	44
빨간색	31	41	보라색	35	45
초록색	32	42	청록색	36	46
갈색	33	43	하얀색	37	47

프롬프트 특수 기능 번호

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

06 프롬프트 설정

- 컬러 프롬프트 설정 예

① 파란색으로 설정하기

```
[park 09:47:30] 854$ PS1="₩e[34mLinux $ ₩e[0;0m"
Linux $
```

② 파란색의 볼드로 설정하기

```
Linux $ PS1="₩e[34;1mLinux $ ₩e[0;0m"
Linux $
```

③ 밑줄 친 빨간색으로 설정하기

```
Linux $  PS1="₩e[31;4mLinux $₩e[0;0m"
Linux $
```

④ 배경은 갈색, 글자는 보라색, 프롬프트는 '사용자 이름@호스트 이름 $ '로 설정하기

```
Linux $ PS1="₩e[33;45;7m₩u@₩h $ ₩e[0;0m"
park@localhost $
```

07 환경 설정 파일

환경설정 파일

- 사용자가 로그인할 때마다 자동으로 실행되는 명령을 저장한 것이 환경 설정 파일
- 시스템 환경 설정 파일과 사용자 환경 설정 파일이 있음
- 셸마다 다른 이름의 파일을 사용

시스템 환경 설정 파일

- 시스템을 사용하는 전체 사용자의 공통 환경을 설정하는 파일

배시 셸의 시스템 환경 설정 파일

파일	기능
/etc/profile	• 시스템 공통으로 적용되는 환경 변수를 설정한다. PATH : 기본 명령 경로 설정 USER, LOGNAME : 사용자 UID와 이름 설정 HOSTNAME : 호스트 이름 설정 HISTSIZE : 히스토리 크기 설정 MAIL : 이메일 설정 • 기본 접근 권한을 설정한다(5장 참조). • /etc/profile.d/*.sh를 실행한다.
/etc/bashrc	• 시스템 공통으로 적용되는 함수와 에일리어스를 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 파일**

```
[user1@localhost ch6]$ more /etc/profile
# /etc/profile

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

# It's NOT a good idea to change this file unless you know what you
# are doing. It's much better to create a custom.sh shell script in
# /etc/profile.d/ to make custom changes to your environment, as this
# will prevent the need for merging in future updates.

pathmunge () {
    case ":${PATH}:" in
        *:"$1":*)
            ;;
        *)
```

07 환경 설정 파일

사용자 환경 설정 파일

- 각 사용자의 홈 디렉터리에 숨김 파일로 생성
- 사용자가 내용을 수정하고 관리 가능

| 배시 셸의 사용자 환경 설정 파일

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

07 환경 설정 파일

- 사용자 환경 설정 파일 예

```
[user1@localhost ~]$ cat .bash_profile
# .bash_profile

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

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
[user1@localhost ~]$
```

07 환경 설정 파일

- 사용자 환경 설정 파일 만들기
  - vi로 .bashrc 파일 수정

```
# .bashrc

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

# Uncomment the following line if you don't like systemctl's auto-paging feature
:
# export SYSTEMD_PAGER=

# User specific aliases and functions
alias rm='rm -i'
alias ls='ls -F'
```

- 사용자 환경 설정 파일 적용하기

```
[user1@localhost ~]$ source .bashrc
[user1@localhost ~]$
[user1@localhost ~]$
[user1@localhost ~]$ ls
linux/    ts1111   tww    다운로드/  바탕화면/  사진/  음악/
services  tslslsl  공개/  문서/     비디오/    서식/
[user1@localhost ~]$
[user1@localhost ~]$
[user1@localhost ~]$ rm tww
rm: remove 일반 파일 `tww'? y
[user1@localhost ~]$
```

07 환경 설정 파일

- 다른 셸의 환경 설정 파일

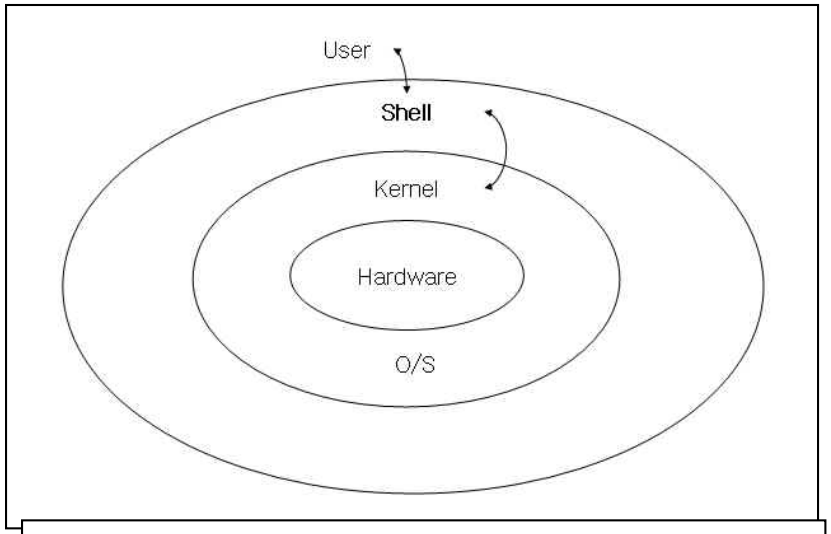
다른 셸의 환경 설정 파일

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

08. Shell Scripts Programming

1. Shell이란

셸은 운영체제를 둘러싸고 입력 받는 명령어를 실행시키는 명령어 해석기로 사용자가 입력한 명령을 해석해서 커널에게 전달하고, 커널의 처리결과를 사용자에게 전달하는 역할을 하며 프로그래밍 언어로서의 특성을 가지고 있다. 최초의 셸은 UNIX 연구 개발 초기 UNIX 파일시스템 설계의 일부분으로 Ken Thompson에 의해 만들어졌으며, AT&T에서 Steven Bourne에 의해 Bourne Shell이 만들어졌다. 리눅스에서의 기본적으로 사용하는 셸은 Bash Shell로 Bourne Again Shell이라는 뜻을 가지고 있다.



셸과 커널과의 관계

08. Shell Scripts Programming

2. Bash(Bourne Again Shell)

➢ Brian Fox 외 몇몇 사람에 의해 1989년에 개발되었으며, FSF에서 내어 놓은 일군의 GNU이다.

➢ Bash의 최종 목표는 IEEE POSIX Shell과 도구 명세에 호환되도록 하는 것이다.

➢ Korn Shell과 C Shell의 유용한 특징들을 통합하고 있으며, 셸 프로그래밍 언어에 있어서는 Bourne Shell과 호환이 된다.

➢ 상호 연결성, 명령어 라인 편집, 완료, 재실행을 지원하는데 매우 탁월한 성능을 나타낸다.

	명칭	실행파일	개발자	개발연도
Bourne Shell 계열	Bourne Shell	/bin/sh	Steven Bourne	1979년
	Korn Shell	/bin/ksh	David Korn	1986년
	Bash	/bin/Bash	Brian Fox 외	1989년
	Z Shell	/bin/zsh	Paul Falstad	1990년
C Shell 계열	C Shell	/bin/csh	Bill Joy	1981년
	TC Shell	/bin/tcsh	Ken Greer 외	1982년

08. Shell Scripts Programming

3. 셸의 활용 및 기능

❶셸의 확인과 변경

- 리눅스에서 지원하고 있는 셸들은 /etc/shells 파일에 그 경로가 설정.
- 리눅스 계정을 가지고 있는 사용자 정보는 /etc/passwd 파일에 모두 저장.
- 사용자 정보는 7개의 필드로 구성 되어 있으며, 마지막 필드에 사용자가 사용하는 쉘이 설정.
- 잠시 사용하고자 하는 셸을 변경하고 싶을 때는 사용하고자 하는 셸을 프롬프트에서 실행.

❷ 셸의 기능

- 셸은 키보드와 화면을 통해서 사용자와 대화하는 인터페이스 기능을 제공
- 강력한 프로그래밍 언어 기능을 지원
- 사용자의 작업 환경을 사용자의 요구사항에 맞추어 설정할 수 있는 기능 제공
- 개인이나 그룹의 스타일이 반영되는 환경을 조성할 수 있다.

❸ 셸의 다양한 기능

- 명령 완성 기능(입력할 파일명이나 디렉토리 또는 명령어의 길이가 너무 긴 경우 명령어의 일부만 입력하고 Tab 키를 누르게 되면, 나머지 입력되지 않은 부분이 완성되어 생성)

08. Shell Scripts Programming

❹ 방향키로 History 사용하기

➢ 로그인을 한 사용자가 지금까지 사용했던 명령어를 보려면, history 명령을 사용

➢ 위, 아래 방향키를 사용함으로써 사용했던 명령어를 찾아낸다.

➢ ~/.bash_history 파일에 저장되어 있으며, ~/.bashrc 파일의 HISTSIZE 변수로 설정할 수 있다.

❺ 마우스를 사용한 텍스트 복사

➢ 멀티콘솔은 X윈도우를 사용하는 경우 여러 개의 터미널을 띄워 동시에 멀티 태스킹을 지원하는 것을 의미

➢ 하나의 터미널에서 다른 터미널로 복사하고자 할 때 마우스로 해당 부분을 드래그하고 복사되기 원하는 위치로 가서 3버튼 마우스인 경우에는 가운데 버튼을 누르고, 2버튼 마우스인 경우에는 두 개 버튼을 동시에 눌러 복사한다.

❻ 와일드 카드의 지원

➢ ? 또는 * 를 사용하는 와일드 카드를 지원한다.

➢ ? : 임의의 한 문자 의미

➢ * : 문자 수와 상관 없이 임의의 문자를 의미

08. Shell Scripts Programming

4. 셸의 환경설정

❶ 환경 설정 파일

➢ 셸 환경 설정 파일은 기본적으로 사용자의 홈 디렉토리 안에 위치

➢새로운 사용자를 등록하면 /etc/skel 디렉토리에 기본값으로 저장되어 있는 파일들을 사용자의 기본 홈 디렉토리로 복사하여 생성된다.

파 일	설 명
.bashrc	셸을 위한 셸 스크립트로 서브 셸, 즉 로그인 셸이 실행될 때 명령과 프로그램 구조로 구성할 수 있다. 새로운 셸이 실행될 때마다 실행
.bash_profile	로그인할 때 읽어 들이는 설정 파일로 주요 설정 내용은 검색경로, 터미널 종류, 환경변수 등을 설정하며 로그인 시점에서 실행시키고 싶은 명령, 시스템에 대한 정보를 보여주는 명령 등을 수행
.bash_logout	로그인 셸이 종료되면서 읽어 들인다.
/etc/profile	시스템 전역에 영향을 미치는 설정 파일
/etc/bashrc	시스템 전역에 영향을 미치고 새로운 셸이 실행될 때 마다 진행

08. Shell Scripts Programming

1). bash 셸 내장 명령어

셸 스크립트는 반복되는 일련의 작업을 셸 스크립트로 만들어 사용, 시스템 관리를 자동화 할 수 있다.

명 령 어	기 능
:	Null 명령어, 0값(참) 돌려주기만 하는 명령어
.	현재의 셸 환경을 실행
break	for, while, until 등의 루프를 벗어나는 명령어
continue	for, while, until 등의 루프에서 이 후 명령어를 건너뛰고 계속 진행
echo	주어진 인수를 표준출력
eval	인수의 실제 값 구하기
exec	현재의 셸을 사용하여 인수로 주어진 명령어를 실행
exit	셸을 종료, 루프 벗어나기
export	지역변수를 환경변수로 변환
expr	표현식의 값을 구한다
printf	포맷을 정하여 출력
read	표준입력장치로 인수입력
readonly	변수를 읽기 전용으로 만들기
return	. 나 소스로 실행된 스크립트의 실행이나 함수를 종료

08. Shell Scripts Programming

1). bash 셸의 내장 명령어

명 령 어	기 능
set	변수설정
test 또는 [ ]	조건이 참이면 0값을 아니면 다른 값을 리턴
trap	실행중인 프로세스가 지정된 신호를 받으면 특별한 동작을 한다
umask	파일과 디렉토리 생성 시 퍼미션의 기본 설정 값을 지정
unset	생성된 변수를 삭제
wait	자식 프로세스가 종료될 때 까지 기다린다.
'명령어'	명령어를 실행하여 결과에 대입

예1) **help 명령어로 내장 명령어 도움말 얻기**

[root@localhost root]# help
[root@localhost root]# help for

예2) 주요 내장 명령어 이해하기

❶ : (콜론)-Null 명령어로 0값을 돌려주기만 하는 명령어로 명령어가 있어야 할 곳을 대신 채워주는 역할

❷ . (도트)-현재의 셸 환경을 가지고 실행

예) . /etc/rc.d/rc.local.mine ; /etc/rc.d/rc.local.mine 파일을 실행

❸ break : for, while, until 등의 루프를 벗어나는 명령어로 루프 중단

❹ continue : for, while, until 등의 루프에서 이후의 명령어를 건너뛰고 루프 계속 진행

❺ echo : 인수를 표준출력으로 출력

08. Shell Scripts Programming

2). 주요 내장 명령어 이해하기

❺ echo : 인수를 표준출력으로 출력 –e 옵션과 같이 사용하는 escape문자

문 자	기 능
₩a	경고음
₩b	백스페이스
₩c	줄바꾸지 않고 붙여서 출력
₩f	폼피드(프린터에서 다음 페이지, 터미널에서 다음 화면)
₩n	개행문자
₩r	캐리지 리턴(다음 행)
₩t	수평탭
₩₩	백슬래쉬 표시

❻ exec : 현재의 셸을 다른 셸이나 명령어로 대체하여 실행
❼ exit : 현재 셸을 종료
❽ export : 지역변수를 환경변수(전역) 전환
❾ read : 명령행에서 인수를 입력받음(사용자와 인터렉티브한 작업)
❿ return : .나 소스로 실행된 스크립트의 실행을 종료하거나 함수를 종료

08. Shell Scripts Programming

2. Bash(Bourn Againe Shell)

1)셸의 기능
- Alias 기능(명령어 단축 기능)
  예) # alias ls3 = “ls -la”

- History 기능(상하 화살표 키)
- 연산기능
- JOB 제어 기능
- 자동 이름완성 기능[TAB 키)
- 프롬프트 제어 기능
- 명령어편집 기능
- 확장 명령어

2)셸 명령문 처리방법
  형식 : [프롬프트] 명령어 [옵션…] [인수…]

  예) # ls -la
      # find . / -name “*.txt”

08. Shell Scripts Programming

2. Bash(Bourn Againe Shell)

3)셸의 환경변수

환경변수	기 능	환경변수	기 능
HOME	현재 사용자 홈 디렉토리	PATH	실행파일이 있는 디렉토리 경로
LANG	기본으로 지원되는 언어	PWD	현재 작업 디렉토리
TERM	로그인 터미널 타입	SHELL	로그인 셸
USER	현재 사용자 이름	DISPLAY	X 디스플레이 이름
COLUMNS	현재 터미널의 컬럼 수	LINES	터미널의 라인 수
PS1	프롬프트 변수 1차	PS2	프롬프트 변수 2차
BASH	Bash 셸의 경로	BASH_VERSION	bash 버전
HISFILE	히스토리 파일의 경로	HISTSIZE	히스토리 파일에 저장되는 수
HOSTNAME	호스트의 이름	USERNAME	현재 사용자 이름
LOGNAME	로그인 이름	LS_COLORS	Ls 명령의 확장자 색상 옵션
MAIL	메일을 보관하는 경로	OSTYPE	OS 타입

08. Shell Scripts Programming

3)셸의 환경변수 예제

```
[user1@localhost ~]$ echo $HOSTNAME
localhost.localdomain
[user1@localhost ~]$
[user1@localhost ~]$ echo $HOME
/home/user1
[user1@localhost ~]$
[user1@localhost ~]$ echo $LANG
ko_KR.UTF-8
[user1@localhost ~]$
[user1@localhost ~]$ echo $TERM
xterm-256color
[user1@localhost ~]$
[user1@localhost ~]$ echo $COLUMNS
80
[user1@localhost ~]$
[user1@localhost ~]$ echo $PS1
[₩u@₩h ₩W]₩$
[user1@localhost ~]$
```

```
[user1@localhost ~]$ echo $HISTFILE
/home/user1/.bash_history
[user1@localhost ~]$
[user1@localhost ~]$ echo $LOGNAME
user1
[user1@localhost ~]$
[user1@localhost ~]$ export LOGNAME=gachon
[user1@localhost ~]$ echo $LOGNAME
gachon
[user1@localhost ~]$
```

08. Shell Scripts Programming

2. Bash(Bourn Againe Shell)

4)셸 스크립트를 배우는 이유

- 모든 데몬을 실행하기 위한 명령들이 스크립트 파일이다.
- 예) /sbin/ 스크립트들
- 예) /sbin/httpd

4)셸 스크립트의 작성과 실행

1)작성 : 에디터(vi, gedit 등)를 이용하여 작성

```
[user1@localhost ~]$ vi name.sh
```

```
#! /bin/bash
echo "사용자 홈    : " $HOME
echo "호스트 이름 : " $HOSTNAME
exit 0
```

08. Shell Scripts Programming

4)셸 스크립트의 작성과 실행

1)작성 : 에디터(vi, gedit 등)를 이용하여 작성 후 실행하기

```
[user1@localhost ~]$ ls -la name.sh
-rw-rw-r-- 1 user1 user1 89  4월   4 12:42 name.sh
[user1@localhost ~]$
[user1@localhost ~]$ sh name.sh
사용자 홈    :  /home/user1
호스트 이름 :  localhost.localdomain
[user1@localhost ~]$
```

#. (모든 사용자도 실행 가능하게 하려면 /usr/local/bin 복사 후 속성을 755변경하면 된다)

```
[user1@localhost ~]$ ls -la name.sh
-rw-r--r-- 1 user1 user1 89  4월   4 12:42 name.sh
[user1@localhost ~]$
[user1@localhost ~]$ chmod 755 name.sh
[user1@localhost ~]$
[user1@localhost ~]$ ls -la name.sh
-rwxr-xr-x 1 user1 user1 89  4월   4 12:42 name.sh
[user1@localhost ~]$
```

```
[user1@localhost ~]$ ./name.sh
사용자 홈    :  /home/user1
호스트 이름 :  localhost.localdomain
[user1@localhost ~]$
```

08. Shell Scripts Programming

5) 변수

-변수에 값이 처음 할당되면 자동으로 변수가 생성

-모든 변수가 문자열(string)로 취급(숫자를 넣어도 문자로 취급)

-변수명은 대소문자를 구분(예: $abc, $ABC는 다른 변수)

-변수에 값을 대입할 때 ' =' 좌우공백이 없어야 한다.

```
[user1@localhost ~]$ vartest=gachon
[user1@localhost ~]$
[user1@localhost ~]$ echo vartest
vartest
[user1@localhost ~]$
[user1@localhost ~]$ echo $vartest
gachon
[user1@localhost ~]$
[user1@localhost ~]$ vartest=gachon computer
bash: computer: 명령을 찾을 수 없습니다...
[user1@localhost ~]$
[user1@localhost ~]$ vartest="gachon computer"
[user1@localhost ~]$
[user1@localhost ~]$ echo vartest
vartest
[user1@localhost ~]$
[user1@localhost ~]$ echo $vartest
gachon computer
[user1@localhost ~]$
```

08. Shell Scripts Programming

6) 변수의 입력과 출력

-$ 문자가 들어간 글자를 출력하려면 ‘ ‘로 묵어주거나 ₩를 붙여준다.

-“ ”는 효과없음

- exit 0 = 셸 스크립트는 실행중간에 문제가 생겨도 무조건 성공을 반환하기 때문에 마지막 행에 성공인지 실패인지 여부를 반환 하므로 0은 성공, 1은 비정상 종료를 표시

```
[user1@localhost ~]$ vi var1.sh
```

```
#!/bin/bash
myvar='Hi Gachon students'
echo $myvar
echo "$myvar"
echo 'myvar'
echo ₩$myvar
echo 값 입력:
read myvar
echo "myvar = $myvar"
exit 0
```

```
[user1@localhost ~]$ sh var1.sh
```

```
[user1@localhost ~]$ sh var1.sh
Hi Gachon students
Hi Gachon students
myvar
$myvar
값 입력:
gachon
myvar = gachon
[user1@localhost ~]$
```

08. Shell Scripts Programming

7) 숫자계산

- 변수에 대입된 값은 모두 문자열 취급
- 변수의 값을 연산(가감승제)하려면 expr을 사용해야 한다.
- (`) ` 역따옴표로 묶어 주어야하며, 모든 연산자와 숫자, 변수, 기호 사이에 space가 있어야 한다.
- = 사이에 space가 있으면 안되며, 괄호와 연산자 사용 시 그 앞에 ₩ 사용해야 한다.
- +와 – 와 달리 (*) 에는 예외적으로 ₩를 붙여야 한다.

```
[user1@localhost ~]$ vi number.sh
```

```
 1 #!/bin/sh
 2 num1=100
 3 num2=`expr $num1 + 30`
 4 echo $num2
 5 num3=`expr $num1 + 200`
 6 echo $num3
 7 num4=`expr ₩( $num1 + 200 ₩) / 10 ₩* 2`
 8 echo $num4
 9 exit 0
10
```

```
[user1@localhost ~]$ sh number.sh
130
300
60
```

08. Shell Scripts Programming

8) 파라미터(Parameter) 변수
 - 파라미터 변수는 $0, $1, $2, $3 . . . 형태로 구성, 전체 파라미터 $*로 표시

예: # yum –y install xinetd

명 령 어	yum	–y	install	xinetd
파라미터변수	$0	$1	$2	$3

```
[user1@localhost ~]$ vi parameter.sh
1 #!/bin/sh
2 echo " 실행할 파일명은 <$0>이다"
3 echo " 첫번째 파라미터는 <$1>이고, 두번째 파라미터는 <$2>이다"
4 echo " 전체 파라미터는 <$*>이다"
5 exit 0
```

```
[user1@localhost ~]$ sh parameter.sh yum -y install xinetd
 실행할 파일명은 <parameter.sh>이다
 첫번째 파라미터는 <yum>이고, 두번째 파라미터는 <-y>이다
 전체 파라미터는 <yum -y install xinetd>이다
```

08. Shell Scripts Programming

9) If 문과 case 문

(1) 기본 if문 – [조건]의 사이의 단어에는 공백이 있어야 한다. (=과 !=), 괄호대신 test 사용가능
[ ] 괄호 앞 뒤에 한칸 이상의 공백이 필요

```
if    [조건]
then
    true-statement
fi
```

```
[user1@localhost ~]$ vi if1.sh
```

```
1 #!/bin/bash
2 if      [ "Gachon" = "Gachon" ]
3         then
4         echo    "true statement"
5 fi
6 exit 0
7
```

```
[user1@localhost ~]$ sh if1.sh
true statement
[user1@localhost ~]$
```

08. Shell Scripts Programming

9) If 문과 case 문

(2) if ~ else 문 – [조건]의 결과가 True, False로 구분(참고 : “ “ 사이에 공백도 조심.

```
if    [조건]
then
    true-statement
else
    false-statement
fi
```

```
[user1@localhost ~]$ vi if2.sh

1 #!/bin/bash
2 if      [ "Gachon" = "Gachon" ]
3         then
4                 echo  "참입니다."
5         else
6                 echo  "거짓입니다."
7 fi
8 exit 0

[user1@localhost ~]$ sh if2.sh
참입니다.
[user1@localhost ~]$ _
```

08. Shell Scripts Programming

9) If 문과 case 문

(3) 조건문에서 비교 연산자 – 문자열 끼리 비교하여 참, 거짓

비교 연산자	기 능
“string1”=“string2”	같으면 true
“string1”!=“string2”	같지 않으면 true
-n “string”	not null 문자열이 아니면 true
-z “string”	null 문자열이면 true

(4) 산술비교 연산자

산술 연산자	기 능
수식1 -eq 수식2	수식1 =수식2 true
수식1 -ne 수식2	수식1 not = 수식2 true
수식1 -gt 수식2	수식1 >수식2 true
수식1 -ge 수식2	수식1 >=수식2 true
수식1 -lt 수식2	수식1 < 수식2 true
수식1 -le 수식2	수식1 <=수식2 true
수식!	수식 False이면 true

08. Shell Scripts Programming

9) If 문과 case 문

(3) 조건문에서 비교 연산자, 산술연산자

```
[user1@localhost ~]$ vi if3.sh

1 #!/bin/bash
2 if      [ 100 -eq 200 ]
3         then
4         echo "100과 200은 같다"
5         else
6         echo "100과 200은 다르다"
7 fi
8 exit 0
```

```
[user1@localhost ~]$ sh if3.sh
100과 200은 다르다
[user1@localhost ~]$
```

08. Shell Scripts Programming

9) If 문과 case 문

(4) 파일관련 조건문 – If문에서 파일처리를 위한 조건

조 건	기 능
-d 파일명	파일이 **디렉토리**이면 true
-e 파일명	파일이 **Exist** 이면 true
-f 파일명	파일이 **일반파일**이면 true
-g 파일명	파일이 **set-group-id**가 설정되면 true
-r 파일명	파일이 **읽기 가능**하면 true
-s 파일명	파일의 **크기가 0**이 아니면 true
-u 파일명	파일이 **set-user-id**가 설정되면 true
-w 파일명	파일이 **쓰기가능**상태이면 true
-x 파일명	파일이 **실행가능상태**이면 true

08. Shell Scripts Programming

9) If 문과 case 문

(4) 파일관련 조건문 – If문에서 파일처리를 위한 조건(예제)

```
[user1@localhost ~]$ vi if4.sh
```

```
1 #!/bin/bash
2 fname=/lib/systemd/system/httpd.service
3 if      [ -f $fname ]
4         then
5         head -5 $fname
6         else
7         echo "웹서버가 설치 되어 있지 않습니다."
8 fi
9 exit 0
```

```
[user1@localhost ~]$ sh if4.sh
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)
[user1@localhost ~]$
```

08. Shell Scripts Programming

9) If 문과 case 문

(5) case ~ esac문 – 2개 이상 분기할 때 (다중분기) – 첫 번째 파라미터 변수 $1의 값에 따라서 분기

```
[user1@localhost ~]$ vi case1.sh
 1 #!/bin/bash
 2 case    "$1"    in
 3         start)
 4                 echo " 사작 ~~~~ ";;
 5         stop)
 6                 echo " 중지 ~~~~ ";;
 7         restart)
 8                 echo " 재시작 ~~~";;
 9         *)
10                 echo " 기타 등등~~";;
11 esac
12 exit 0
```

```
[user1@localhost ~]$ sh case1.sh
 기타 등등~~
[user1@localhost ~]$ sh case1.sh start
 사작 ~~~~
[user1@localhost ~]$ sh case1.sh stop
 중지 ~~~~
[user1@localhost ~]$ sh case1.sh restart
 재시작 ~~~
[user1@localhost ~]$ sh case1.sh 1234
 기타 등등~~
[user1@localhost ~]$
```

08. Shell Scripts Programming

9) If 문과 case 문

(5) case ~ esac문 – 2개 이상 분기할 때 (다중분기)

실행할 문장이 더 없을 때는 반드시 ; ; 입력한다.

exit 1은 정상적인 종료가 아니므로

line6 : 다음 실행 라인이 더 있으므로 ;;하지 않음.

line8 : 앞에 n 또는 N이 있으면 모든 단어를 인정한다.

line12: 정상적인 종료가 아니므로 exit 1로 종료

```
[user1@localhost ~]$ vi case1.sh
```

```
 1 #!/bin/bash
 2 echo "리눅스 재미 있나요? (yes / no)"
 3 read answer
 4 case    $answer in
 5         yes | y | Y | Yes | YES)
 6                 echo " 다행입니다 "
 7                 echo " 더욱 열심히 하세요 ";;
 8         [nN]*)
 9                 echo " 안타갑네요 ";;
10         *)
11                 echo " yes 아니면 no만 입력해야 합니다"
12                 exit 1;;
13 esac
14 exit 0
```

```
[user1@localhost ~]$ sh case1.sh
리눅스 재미 있나요? (yes / no)
y
 다행입니다
 더욱 열심히 하세요
[user1@localhost ~]$ sh case1.sh
리눅스 재미 있나요? (yes / no)
n
 안타갑네요
[user1@localhost ~]$ sh case1.sh
리눅스 재미 있나요? (yes / no)
1234
 yes 아니면 no만 입력해야 합니다
[user1@localhost ~]$ sh case1.sh
리눅스 재미 있나요? (yes / no)
Y
 다행입니다
 더욱 열심히 하세요
[user1@localhost ~]$ sh case1.sh
리눅스 재미 있나요? (yes / no)
N
 안타갑네요
[user1@localhost ~]$ sh case1.sh
리눅스 재미 있나요? (yes / no)
YES
 다행입니다
 더욱 열심히 하세요
```

08. Shell Scripts Programming

9) If 문과 case 문

(6) AND, OR 관계연산자

- and = -a 또는 &&, or = -o 또는 || -a나 -o 는 테스트 문 안에서 사용될 수 있는데 괄호 등으 특수문자 앞에는 역슬레쉬(₩) 붙여줘야 한다.
- If [₩(-f ₩$fname ₩) -a ₩(-s ₩$fname ₩)] ; then

*. (-f =일반파일, -s = 크기가 0가 아닌 파일) - root 에서 실행하면 결과가 나온다.

```
[user1@localhost ~]$ vi andor.sh
```

```
1 #!/bin/bash
2 echo " 보고싶은 파일명을 입력하세요"
3 read fname
4 if      [ -f $fname ] && [ -s $fname ] ; then
5         head -5 $fname
6 else
7         echo " 파이이 없거나 크가가 0입니다."
8 fi
9 exit 0
```

```
[user1@localhost ~]$ sh andor.sh
 보고싶은 파일명을 입력하세요
/lib/systemd/system/httpd.service
[Unit]
Description=The Apache HTTP Server
After=network.target remote-fs.target nss-lookup.target
Documentation=man:httpd(8)
Documentation=man:apachectl(8)
[user1@localhost ~]$
[user1@localhost ~]$ sh andor.sh
 보고싶은 파일명을 입력하세요
abcd
 파이이 없거나 크가가 0입니다.
[user1@localhost ~]$ _
```

08. Shell Scripts Programming

10) 반복문

(1) for 문

```
for 변수  in   값1   값2   값3  ….
do
        반복할 문장
done
```

```
[user1@localhost ~]$ vi for1.sh
```

```
1 #!/bin/bash
2 sum=0
3 for i in 1 2 3 4 5 6 7 8 9 10
4         do
5                 sum=`expr $sum + $i`
6         done
7         echo    " 1부터  10까지의  합  :  " $sum
8 exit 0
```

```
[user1@localhost ~]$ sh for1.sh
 1부터  10까지의  합  :  55
[user1@localhost ~]$
```

08. Shell Scripts Programming

10) 반복문

예) 현재 디렉토리 위치에서 *.sh로 끝나는 셸 스크립트 파일들의 파일명과 파일의 내용을 앞에서 3줄씩 화면에 표시하는 스크립트를 작성하세요.

```
[user1@localhost ~]$ vi for2.sh

1 #!/bin/bash
2 for fname in $(ls *.sh)
3         do
4                 echo " ----------$ fname ------------"
5                 head -3 $fname
6 done
7 exit 0
8
```

```
[user1@localhost ~]$ sh for2.sh
 ----------$ fname ------------
#!/bin/bash
echo " 보고싶은 파일명을 입력하세요"
read fname
 ----------$ fname ------------
#!/bin/bash
case    "$1"    in
        start)
 ----------$ fname ------------
#!/bin/bash
case    "$1"    in
        start)
 ----------$ fname ------------
#!/bin/bash
sum=0
for i in 1 2 3 4 5 6 7 8 9 10
 ----------$ fname ------------
#!/bin/bash
for fname in $(ls *.sh)
        do
 ----------$ fname ------------
#!/bin/bash
if      [ "Gachon" = "Gachon" ]
        then
```

08. Shell Scripts Programming

10) 반복문

(2) While 문

조건식이 참인 동안 계속 반복 – 조건식에 [1] 또는 [:] 또는 while [true]이 오면 항상 참이다.

무한루프 벗어나기 : Ctrl + C

```
[user1@localhost ~]$ vi while1.sh
1 #!/bin/bash
2 while   [ 1 ]
3         do
4                 echo " Welcome to CentOS7 World "
5         done
6 exit 0
7
```

```
[user1@localhost ~]$ sh while1.sh
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World
 Welcome to CentOS7 World ^C
[user1@localhost ~]$
```

08. Shell Scripts Programming

10) 반복문

(2) While 문

조건식이 참인 동안 계속 반복 – 조건식에 [1] 또는 [:] 또는 while [true]이 오면 항상 참이다.

무한루프 벗어나기 : Ctrl + C

```
[user1@localhost ~]$ vi while2.sh
 1 #!/bin/bash
 2 sum=0
 3         i=1
 4         while [ $i -le 10 ]
 5         do
 6                 sum=`expr   $sum + $i`
 7                 i=`expr $i + 1 `
 8         done
 9                 echo "1부터  10까지의  합:  " $sum
10 exit 0
```

```
[user1@localhost ~]$ sh while2.sh
1부터  10까지의  합:  55
[user1@localhost ~]$
```

08. Shell Scripts Programming

10) 반복문

(2) While 문

[예제]- 비밀번호를 입력 받고 비밀번호가 맞을 때 까지 계속 입력 받는 스크립트를 작성하세요.

```
[user1@localhost ~]$ vi while3.sh

 1 #!/bin/bash
 2 echo "비밀번호를 입력하세요"
 3 read mypass
 4         while [ $mypass != "1234" ]
 5         do
 6                 echo " 비밀번호가 틀립니다. 다시 임력하세요"
 7                 read mypass
 8         done
 9                 echo "환영합니다"
10 exit 0

[user1@localhost ~]$ sh while3.sh
비밀번호를 입력하세요
123444
 비밀번호가 틀립니다. 다시 임력하세요
1234
환영합니다
[user1@localhost ~]$
```

08. Shell Scripts Programming

10) 반복문

(4) unitl 문 (조건이 참이 될 때까지 반복 – 거짓인 동안 만 반복)

[예제]- 1에서 10까지 합을 구하는 스크립트를 until 문으로 작성하세요.

```
[user1@localhost ~]$ vi until.sh

 1 #!/bin/bash
 2 sum=0
 3 i=1
 4 until   [ $i -gt 10 ]
 5         do
 6                 sum=`expr   $sum + $i`
 7                 i=`expr $i + 1`
 8         done
 9                 echo " 1부터  10까지의  합  "
10 exit 0

[user1@localhost ~]$ sh until.sh
 1부터  10까지의  합
[user1@localhost ~]$
```

08. Shell Scripts Programming

10) 반복문

(5) break, continue, exit 문

break – 조건문을 빠져 나갈 때

continue 문은 조건식으로 들어갈 때

exit문은 해당 프로그램 완전 종료 시 사용

```
[user1@localhost ~]$ vi bce.sh
1  #!/bin/bash
2  echo " 무한반복 입력을 시작합니다.(b:break, c:continue, e:exit)"
3          while [ 1 ] ; do
4          read input
5                  case $input in
6                          b | B)
7                          break;;
8                          c | C)
9                          echo " c를 누르면 while의 조건으로 돌아감"
10                         continue;;
11                         e | E)
12                         echo " e를 누르면 프로그램을 완전 종료함"
13                         exit 1;;
14                 esac;
15         done
16         echo "break를 누르면 while문을 빠져 나와 지금 이 문장이 출력 됨"
17 exit 0
```

08. Shell Scripts Programming

10) 반복문

(5) break, continue, exit 문

break – 조건문을 빠져 나갈 때

continue 문은 조건식으로 들어갈 때

exit문은 해당 프로그램 완전 종료 시 사용

```
[user1@localhost ~]$ sh bce.sh
 무한반복 입력을 시작합니다.(b:break, c:continue, e:exit)
b
break를 누르면 while문을 빠져 나와 지금 이 문장이 출력 됨
[user1@localhost ~]$
[user1@localhost ~]$ sh bce.sh
 무한반복 입력을 시작합니다.(b:break, c:continue, e:exit)
c
 c를 누르면 while의 조건으로 돌아감
c
 c를 누르면 while의 조건으로 돌아감

c
 c를 누르면 while의 조건으로 돌아감
b
break를 누르면 while문을 빠져 나와 지금 이 문장이 출력 됨
[user1@localhost ~]$ sh bce.sh
 무한반복 입력을 시작합니다.(b:break, c:continue, e:exit)
e
 e를 누르면 프로그램을 완전 종료함
[user1@localhost ~]$
```

08. Shell Scripts Programming

11) 사용자 정의 함수

사용자가 직접함수를 작성하고 호출할 수 있다.

```
함수 이름 {                          ->함수 정의
        내용들 ….
}
함수이름                             -> 함수를 호출
```

```
[user1@localhost ~]$ vi func1.sh
```

```
1 #!/bin/bash
2 myFunction () {
3         echo " 함수 안으로 들어 왔음"
4         return
5 }
6 echo " 프로그램을 시작합니다."
7 myFunction
8         echo " 프로그램을 종료합니다"
9 exit 0
10
```

```
[user1@localhost ~]$ sh func1.sh
 프로그램을 시작합니다.
 함수 안으로 들어 왔음
 프로그램을 종료합니다
```

08. Shell Scripts Programming

11) 사용자 정의 함수

사용자가 직접함수를 작성하고 호출할 수 있다.

```
함수 이름 {                          ->함수 정의
        내용들 ….
}
함수이름                             -> 함수를 호출
```

```
[user1@localhost ~]$ vi func2.sh
```

```
1 #!/bin/bash
2 sum () {
3         echo `expr $1 + $2`
4 }
5 echo " 10 더하기 20을 계산합니다."
6 sum 10 20
7 exit 0
8
```

```
[user1@localhost ~]$ sh func2.sh
 10 더하기 20을 계산합니다.
30
[user1@localhost ~]$
```

08. Shell Scripts Programming

13) Eval

문자열을 명령문으로 인식하고 실행한다.

```
[user1@localhost ~]$ vi eval.sh

  1 #!/bin/bash
  2 str="ls -la .bash_history"
  3 echo $str
  4 eval $str
  5 exit 0
```

```
[user1@localhost ~]$ sh eval.sh
ls -la .bash_history
-rw------- 1 user1 user1 10566  4월   4 12:20 .bash_history
[user1@localhost ~]$
```

08. Shell Scripts Programming

14) export

외부 변수(전역 변수)로 선언하여 다른 프로그램에서도 변수로 사용

설명 : exp1.sh line2~3 : var1, var2 변수 출력, var2는 exp2.sh에서 외부 변수로 선언

exp2.sh line 2 : var1에 값을 넣는다. 일반 변수로 exp2.sh내에서 만 사용된다.

즉, exp1.sh의 var1과는 우연히 변수명이 같을 뿐 다른 변수다.

```
[user1@localhost ~]$ vi exp1.sh
   1 #!/bin/bash
   2 echo $var1
   3 echo $var2
   4 exit 0

[user1@localhost ~]$ sh exp1.sh

[user1@localhost ~]$
```

```
[user1@localhost ~]$ vi exp1.sh
   1 #!/bin/bash
   2 var1="지역변수"
   3 export var2="외부변수"
   4 sh exp1.sh
   5 exit 0

[user1@localhost ~]$ sh exp2.sh

외부변수
[user1@localhost ~]$
```

08. Shell Scripts Programming

15) printf

C언어의 printf 문과 유사

- printf “%5.2f ₩n₩n ₩t %s ₩n
- %5.2f = 실수로 5자리 소수점 2째 자리 출력 , ₩n = 줄 바꿈, ₩t =Tab, %s = string(문자열) 출력
- 문자열 출력 시 “$var2”-중간에 공백이 있으므로 “ “로 묶어 주어야 한다.

```
[user1@localhost ~]$ vi printf.sh
```

```
1 #!/bin/bash
2 var1=100.5
3 var2="가천대학교 컴퓨터공학과"
4 printf "%5.2f ₩n₩n ₩t %s ₩n" $var1 "$var2"
5 exit 0
```

```
[user1@localhost ~]$ sh printf.sh
100.50

        가천대학교 컴퓨터공학과
[user1@localhost ~]$
```

08. Shell Scripts Programming

16) set명령과 $ (명령어)

리눅스 명령어를 결과로 사용하기 위해서는 $(명령어) 사용해야 한다. 결과를 파라미터로 사용하고자 할 때는 set명령어를 같이 사용한다.(일,월,화,수,목,금,토의 순서)

```
[user1@localhost ~]$ vi set.sh
```

```
1 #!/bin/bash
2 echo "오늘 날짜는 $(date)입니다."
3 set $(date)
4 echo " 오늘은 $4 요일입니다."
5 exit 0
```

```
[user1@localhost ~]$ sh set.sh
오늘 날짜는 2019. 04. 04. (목) 17:34:26 KST입니다.
 오늘은 (목) 요일입니다.
[user1@localhost ~]$
```

08. Shell Scripts Programming

17. 셸 프로그래밍(bash : Bourne Again Shell)

예제1: 화면에 Hello linux world 출력하기

❶ vi 에디터로 hello.sh 파일을 만든다

❷ 파일의 내용은

```
# !/bin/bash
echo Hello linux world
```

설명 : 1)이 파일을 실행하기 위한 프로그램의 경로
2)표준출력장치에 Hello linux world 표시

❸ 실행속성 부여

```
# chmod +x hello.sh
```

❹ 실행

```
# ./hello.sh
```

09.보고서

다음을 설명하세요.

1. 셸의 세가지 주요 기능을 설명하세요.
2. 표준 입출력 장치가 무엇인지 설명하세요.
3. > 사용 시 기존 파일의 내용을 겹쳐 쓸 수 있다. 방지하기 위한 방법을 설명하세요.
4. 특수문자 “ “와 ‘ ‘의 차이점과 사용 예를 설명하세요.
5. 전체 환경변수를 출력하는 **env**명령의 결과에서 특정환경변수(예:**SHELL**)만 선택하서 출력하는 방법을 설명하세요.
6. 현재 디렉토리에 있는 모든 파일의 상세정보는 **ls.out** 파일에 , 오류 메시지는 **ls.err**파일에 저장하는 방법을 설명하세요.