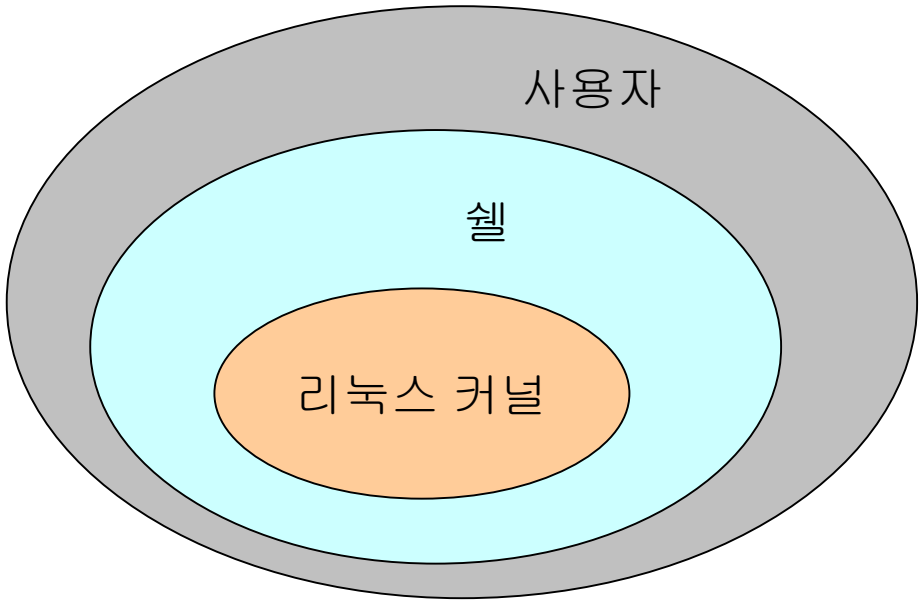
리눅스
쉘(shell) 개요

쉘 (shell) 이란?

- 사용자의 명령어를 해석하여 수행하는 사용자 인터페이스 프로그램
  - 명령어 해석
  - 프로그래밍 언어 기능 : 쉘 프로그래밍



쉘 종류

- 종류
  - Bourne shell (sh)
  - Bourne Again shell (bash) : 리눅스의 기본 쉘

쉘	해당 파일
Bourne	/bin/sh
Bourne Again	/bin/bash

  - Korn shell (ksh)
  - C shell (csh)
  - TC shell (tcsh)

사용하는 쉘의 확인

- 현재 사용 중인 쉘의 확인

  echo **$SHELL**

  cat /etc/passwd

/etc/passwd : 사용자 계정 파일
- login name
- password
- shell
- home directory 등의 정보

- 시스템에서 사용가능한 쉘의 확인

  cat /etc/shells

쉘의 변경

- 사용 쉘의 변경

$ chsh

Password:

Changing the login shell for ysryu

Login shell [/bin/bash]: **/bin/sh**

Shell changed

$ echo $SHELL

변경여부 확인

$ logout

Logout 하고 다시 login

$ _

$ echo $SHELL

/bin/sh

홈 디렉토리

- 홈 디렉토리 (home directory)
  - 사용자가 로그인했을 때 처음으로 위치하는 곳으로
  - 사용자 개인별로 작업을 하는 장소
  - 시스템 관리자가 사용자의 계정을 만들때, 사용자 별로 홈 디렉토리를 만들고 지정해준다.
  - 사용자는 자신의 홈 디렉토리에서 파일을 읽기, 쓰기, 디렉토리 만들기 등을 할 수 있지만, 일반적으로 다른 디렉토리의 내용은 읽기만 할 수 있다. (사용자간 정보 보호)

- 홈 디렉토리 알아보기

```
$ echo $HOME

$ cat /etc/passwd
```

홈 디렉토리

- 틸드(~)의 사용
  - 사용자명 앞에 틸드 문자를 사용하면 그 사용자의 홈 디렉토리를 의미한다.

    예)

    ```
    cd   ~ysryu                (ysryu 사용자의 홈 디렉토리로 이동)
    cat  ~ysryu/sample.c       (ysryu 사용자의 홈 디렉토리에 있는
                                sample.c 파일 보기)
    ```

  - 내 홈 디렉토리는 ~/

    예)

    ```
    cat   ~/sample.c        (내 홈 디렉토리의 sample.c 파일 보기)
    cd ~/        cd ~       cd     (내 홈 디렉토리로 이동)
    ```

쉘의 환경 설정 파일

- 쉘 환경 설정 파일
  - 로그인해서 쉘이 시작될 때 (또는 로그인 이후 쉘을 실행했을 때), 쉘은 정해진 디렉토리에서 환경 설정 파일을 찾고,
  - 파일이 있으면 파일 내용을 읽어 명령들을 실행한다.

Bash

- 개요
- 환경 설정 파일
- 내부 명령어(Built-in command)
- 쉘 변수(Shell variable)
- 표준 입출력 재지정(Standard input/output redirection)
- 메타 문자(Meta character)

- 참고문헌 : “배시 셸 시작하기”, 한빛미디어

Bash 개요

- Bash (Bourne Again Shell)
- /bin/bash
- bourne shell의 기능을 가지며, POSIX shell (IEEE WG 1003.2)과 호환이 되도록 GNU에서 개발
- Korn shell과 C shell의 유용한 기능 일부를 채택

환경 설정 파일

- login하여 bash가 처음 시작할 때(login shell), 다음 스크립트 파일들을 찾고, 있다면 파일의 내용을 수행하여 환경을 설정한다.
  ① /etc/profile
  ② ~/.bash_profile *or* ~/.bash_login *or* ~/.profile
  ③ ~/.bashrc
  ④ /etc/bashrc

$ /bin/bash ↵

- login한 상태에서 bash를 수행하면, 다음 파일을 찾아서 수행함
  – ~/.bashrc

- login shell이 logout할 때 다음 파일을 찾아서 수행함
  – ~/.bash_logout

(1) ~/ 는 로그인 사용자의 홈 디렉토리를 의미함
(2) login shell : login할 때 실행된 쉘

기타 환경 설정 파일

- 히스토리 (history) 관련 파일
  - ~/.bash_history

- /etc/skel
  - root 사용자가 일반 사용자 계정을 생성할 때, 홈 디렉토리에 복사될 파일을 두는 디렉토리
  - 보통, .bash_profile, .bashrc, .bash_logout 파일이 있다.

Built-in command (내부 명령어)

- 쉘 프로그램 자체가 내부적으로 처리하는 명령어

echo	문자열의 출력
read	사용자로부터 값을 읽음
:	아무 액션도 없으며 참값을 반환
.	지정한 파일로부터 읽고 실행 (현재 쉘 환경에서 실행)
source	. 과 같음
alias	명령어에 대한 별명(alias)을 지정
bg	특정 프로세스를 백그라운드로 실행
cd	디렉토리 이동
exit	종료
logout	

Built-in command

hash	입력한 명령어의 경로를 해쉬 테이블에 저장
history	이전에 사용한 명령어를 보여줌
export	환경변수를 설정 (쉘 변수를 환경에 추가)
pwd	현재 디렉토리 출력
set	변수 설정
unset	변수 초기화
printf	정형화된 데이터의 출력

continue, break, eval, let, test, kill, wait, suspend, exec, jobs, ……

echo

- echo 사용예

```
$  name="mju"          ; name이라는 쉘변수를 생성하고
                         값을 mju로 지정
$  echo  $name         ; name의 값을 출력
 mju
```

echo

- 색깔있는 글자 출력하기
- ANSI Escape code을 사용함
  - ^[[숫자m

^[[입력방법
crtl-v ctrl+Esc [

색상	글자색 코드	배경색 코드
검정색(회색)	30	40
빨간색(밝은 빨강)	31	41
초록색(밝은 초록)	32	42
갈색(노랑색)	33	43
파란색(밝은 파랑)	34	44
기본 화면색	0	
Bold intensity	1	

$ echo “^[[31m안녕하세요^[[0m”

read

- read 사용 예

```
$ read var_value                    ; 키보드 입력을 받아 var_value의 값으로
My name is Yeonseung Ryu
$ echo $var_value
My name is Yeonseung Ryu
```

```
$ read –p “Input Name : “ name_value
Input Name : ysryu
$ echo $name_value
ysryu
```

alias

- alias : 명령어를 다른 이름으로 만듬.

  alias name=command

  alias lf='ls –F'
  alias ls='ls –l'
  alias rm='rm –i'

- unalias : alias 정의를 삭제

  unalias name

- alias 명령어를 매개변수 없이 수행하면, 정의되어 있는 alias 목록이 출력됨

  $ alias

쉘의 옵션 (option)

- 쉘의 옵션
  - 쉘의 동작을 제어하기 위해 제공되는 기능으로 set 명령어를 사용하여 설정할 수 있다.
  - 예)
    - ignoreeof : ctrl-d 를 사용하여 logout하는 것을 방지함
    - noclobber : >를 사용하여 기존 파일에 덮어 쓰는 것을 방지함
    - noglob : 와일드카드(*, ?)의 확장을 방지함

- set 명령어
  - 옵션을 켤 때 : set –o optionname
  - 옵션을 끌 때 : set +o optionname
  - 옵션들의 상태를 보려면 : set -o

쉘 변수

- 쉘 변수는 **시스템 변수**와 **사용자 정의 변수**가 있다.

- 변수의 값 설정

  변수명=변수값 (주의) =을 붙여서 사용

```
$ PATH=/ysryu/bin
$ echo $PATH
/ysryu/bin
```

- 변수의 값 출력

  echo $변수명

변수 이름의 규칙
- 영문자의 대소문자, 숫자, 밑줄(_)로 구성
- 첫번째 문자는 문자로 시작
- 시스템 변수 사용 금지
- 내부 및 외부 명령어는 변수로 사용할 수 없다
- 대소문자를 구별한다.

쉘 변수

- 시스템 변수
  - 시스템에서 정의한 쉘 변수 (부팅할 때 쉘 설정파일에서 정의함)
  - 대부분 읽기전용으로 변경불가이지만, 어떤 변수들은 사용자가 변경할 수 있다.
  - set, env, printenv 명령어 등으로 확인할 수 있다.

PS1	쉘의 기본 프롬프트
PS2	쉘의 하위 프롬프트
HOME	홈 디렉토리
PATH	명령어의 경로 지정
USER	사용자 이름

쉘 변수

- 시스템 변수 (계속)

SHELL	쉘 프로그램 파일이 위치하는 디렉토리 경로
MAIL	메일 보관 디렉토리 경로
LOGNAME	현재 로그인한 사용자 이름
HOSTNAME	호스트 이름
HISTFILESIZE	히스토리 파일의 크기
HISTSIZE	히스토리 사용 개수
PWD	현재 디렉토리의 절대 경로
SECONDS	bash을 시작하여 경과된 시간
IGNOREEOF	EOF가 입력되었을 때 쉘의 종료를 제어
UID	사용자의 UID 값
GID	사용자의 GID 값

쉘 변수

- PATH 변수
  - 명령어를 실행할 때 탐색할 디렉토리
  - 콜론(:)으로 여러 디렉토리를 구분하여 나열

  - 예:
    - 기존 PATH 변수에 /home/ysryu/bin을 추가하는 경우

      PATH=$PATH":/home/ysryu/bin"

쉘 변수

- PS1 변수 : 프롬프트 모양

/etc/profile 이나 .bashrc 에
기록하면 로그인할때마다 설정됨

  - \d "요일 달 날짜" 형식의 날짜 표시
  - \H 도메인 이름
  - \h 첫번째 "."까지의 호스트 이름
  - \s 쉘의 이름
  - \u 사용자 이름
  - \w 현재 작업 디렉토리의 절대 경로명
  - \W 현재 작업 디렉토리의 절대 경로명 중 마지막 디렉토리 명
  - \t 24-시간 형식으로 현재 시각 (HH:MM:SS)
  - \T 12-시간 형식으로 현재 시각
  - \@ 12-시간으로 현재 시각, 오전/오후
  - \! 현재 명령어의 히스토리 번호
  - \# 현재 명령어의 번호
  - \$ 현재 UID가 0이면 #를, 아니면 $를 표시

```
$ PS1="[\u@\h  \W]"
[ysryu@sslab bin] _
```

쉘 변수

- 사용자 정의 변수
  - 사용자가 정의하는 변수

```
$ my_name=“ysryu”
$ echo $my_name
ysryu
```

쉘 변수

- 사용자 정의 변수
  - 공백이 있는 변수를 정의할 때는 " 또는 '를 사용하여 묶어준다

```
$ my_name="yeon seung ryu"
$ echo  $my_name
yeon seung ryu
```

- 명령어의 실행 결과를 변수로 지정하려면
  - Back quote (`)를 사용

```
$ result=`ls /etc/passwd`
$ echo  $result
/etc/passwd
```

쉘 변수

- 변수 값의 제거

```
$ my_name=“yeon seung ryu”
$ echo  $my_name
yeon seung ryu
$ unset  my_name
$ echo  $my_name
```

쉘 변수

- export 명령어 : 변수를 환경변수로 만듬

```
$ COLOR=red
$ export COLOR
```

그냥 export를 하면 환경변수를 볼수있다.
$ export

or

```
$ export COLOR=red        ← COLOR를 환경에 추가
$ export –n COLOR         ← COLOR를 환경에서 제거
```

set –a나 set –o allexport를 하면
이후에 정의하는 모든 변수를 환경변수로 만듬

쉘 변수

- 환경(environment)
  - 쉘에 의해 초기화되어 export된 쉘 변수들
  - 쉘의 자식 프로세스(*쉘에 의해 수행되는 프로그램(명령어)*)에게 상속되어 전해짐

- 환경에 변수를 추가하기 위한 두가지 방법 (즉, 환경변수로 만들려면)

  export name=value

  declare –x name=value

- 환경 변수가 아닌 것은 쉘의 자식 프로세스에 상속되지 않는다.

쉘 변수

- 환경의 상속 예

```
$ NAME=YSRYU
$ export NAME2=KDHONG
$ /bin/bash                    ← 자식 프로세스를 만듬
$ echo $NAME                   (상속안됨)
$ echo $NAME2                  (상속됨)
KDHONG
$ exit                         ← 자식 프로세스 종료, 부모에게 돌아감
$ echo $NAME
YSRYU
```

자식 프로세스인 /bin/bash가 수행중

쉘 변수

- 특별 내장 변수 : 쉘이 내부적으로 관리하는 변수

$$	실행중인 프로세스의 PID
$?	마지막에 실행한 명령어의 종료값
$!	마지막 자식 프로세스의 PID
$#	명령어에 전달된 매개변수의 개수
$0	명령어 실행 시에 명령어를 저장하는 변수
$1, $2, ..	명령어 실행 시에 매개변수들을 저장하는 변수
$*	명령어 실행 시에 전달된 매개변수 전체를 하나의 문자열로 가짐
$@	매개변수를 문자열의 목록으로 표시하고 각 인자에 큰 따옴표로 처리
$_	마지막 명령어나 명령어에 전달된 인수값

Meta character

- 메타 문자
  - 쉘이 특수하게 처리하는 문자
  - 메타문자로 취급하지 않게 하려면 문자 앞에 \를 붙여준다

  - Redirection : > < >>
  - 파일 이름 대치 : * ? [ ]
  - 파이프 : |
  - 명령어 대치 : `
  - 명령어 열 : ; || &&
  - 명령어 그룹 : ( )
  - 백그라운드 실행 : &

redirection

- 표준 입력을 파일에서 : <
  $ cat **<** /etc/passwd

- 표준 출력을 파일로 : >
  $ ls **>** dirlist

- 표준 출력을 파일의 끝에 추가(append) : >>
  $ ls **>>** dirlist

- 표준 에러를 파일로 : 2> (정상적인 표준 출력은 모니터로 출력)
  $ ls **2>** errfile

- 표준 에러, 표준 출력을 파일로 : 2>&1 또는 &> 또는 >&
  $ ls **>** dirlist **2>&1**
  $ ls **&>** dirlist

redirection

- 표준 출력, 표준 에러를 모두 안나오게 하려면

  $ ls **&>** /dev/null

- Here document
  - *<< label*

```
$   cat >> msgfile  << .
> This is the text of
> our message.
> .
```

← 이 예제에서는 .(dot)를 label로 사용함

파일 이름 대치 (file name expansion)

globbing이라고도 함

- * : 공백을 포함하는 어떠한 문자열과도 일치
- ? : 어떤 단일 문자와 일치
- [...] : 대괄호 사이의 어떤 하나의 문자와 일치. 문자의 범위는 대시(-) 기호를 사용하여 지정 가능

```
$ ls *.c                ···· ".c" 로 끝나는 모든 것
a.c   b.c    cc.c

$ ls ?.c                ···· ".c" 앞에 하나의 문자가 있는 것
a.c    b.c

$ ls [ac]*              ··· a 또는 c로 시작하는 것
a.c    cc.c

$ ls [A-Za-z]*          ··· A에서 Z 또는 a에서 z사이의 한 문자로 시작하는 것
```

파이프 (pipe)

- 한 프로세스의 표준출력을 다른 프로세스의 표준 입력으로 사용하게 함
  $ 명령어1 | 명령어2

  $ ls | wc

  $ cat /etc/passwd | awk –F: '{ print $1 }' | sort

명령어 대치 (substitution)

- Back quote (`) 또는 $( )로 둘러싸인 명령어는 실행되어 결과값이 출력된다

$ echo the date today is **`date`**

$ echo there are **`who | wc –l`** users on the system

$ echo the date today is **$(date)**

명령어 열

- 세미콜론(;)으로 분리되는 일련의 명령어는 차례로 실행된다.

  $ date ; pwd ; ls …… 3개의 명령어를 순차적으로 실행

- 조건부 명령어 열
  - && 이전 명령어의 종료값이 성공한 경우(0인 경우)만 다음 명령이 실행
  - || 이전 명령의 종료값이 실패한 경우(0이 아닌 경우)만 다음 명령이 실행

  $ gcc myprog.c && a.out

  $ gcc myprog.c || echo compilation failed

명령어 그룹

- 명령어들을 괄호 안에 그룹으로 묶으면, 이 명령들은 서브쉘(sub shell)에서 실행된다.
  - 서브쉘 : 현재 쉘에서 만든 자식 쉘

```
$ (date; ls; pwd)
```

- 명령어들을 브레이스 { }로 묶으면 현재 쉘에서 실행된다.

```
$ { date; ls; pwd; }
```

주의: {}와 문자 사이를 띄어야 함
명령어 다음에 ;을 해야 함

백그라운드 실행

- 명령어, 파이프라인, 명령어 그룹 뒤에 &를 붙이면 서브쉘을 생성하여 백그라운드로 실행한다.
- 백그라운드 처리는 부모쉘과 동시에 실행되며, 키보드를 제어하지 않는다.

```
$ date &
```

- 관련 명령어
  - jobs
  - fg
  - bg
  - kill

인용부호의 사용

- “ “ : 특수문자 및 변수를 사용할 수 있게 해줌
- ‘ ‘ : 특수문자 및 변수 의미를 제거함

```
$ var1=“hello”
$ echo “$var1”        ….. 변수를 사용하게 함
hello

$ echo ‘$var1’        ….. 변수의 의미를 제거하고 일반문자열로 취급
$var1
```

인용부호의 사용

- Example

  $ echo 2 * 3 > 5 is a valid inequality

  $ echo ‘2 * 3 > 5 is a valid inequality’

  $ echo ‘2 * 3 > 5’ is a valid inequality

메타문자 제거

- \ : 뒤따라오는 메타문자의 의미를 제거함 (backslash-escaping)

```
$ echo \$var1          ..... $의 의미를 제거하여 $var를 일반문자열로 취급
$var1
```

```
$ echo 2 \* 3 \> 5 is a valid inequality
```

```
$ echo \"2 \* 3 \> 5\" is a valid inequality
```

```
$ echo This is a text \
> of our message.
```

RETURN의 의미를 무시하게 함

쉘의 2차 프롬프트 (PS2)