Shell: 제대로 사용하기

- Bash 소개
- 명령 기본+
- Redirection
- 명령행 편집
- 사용자 환경 설정

셸 (Shell)

- 사용자가 입력한 명령을 해석하고 실행하는 명령 해석기 (Command Interpreter)
  - 사용자가 처음 수행할 수 있는 특수 프로그램

- UNIX 셸 종류
  - sh: Stephen Bourne, 기본 표준 셸, $ prompt
  - csh: Bill Joy, C와 닮은 꼴, % prompt
  - ksh: David Korn, sh과 호환, $ prompt
  - zsh: Paul Falstad, ksh의 업그레이드
  - bash: Brian Fox, GNU free software, sh과 호환, csh과 ksh의 장점 수용, $ prompt
  - 그 외에도 많음.

셸의 기능

❑ 셸의 공통 기능

- 다수의 내장 명령어 제공
- 메타문자
- 입력/출력/오류의 재지정 (redirection)
- 파이프라인
- 명령어 대치/완성/편집
- 환경 변수와 지역변수
- 후면 처리
- 하위셸(subshell) 생성
- 셸 프로그래밍
- 작업 제어

셸 시작

❑ 로그인

- 계정 생성 시 기본 셸이 지정된다.
  - 로그인 셸 확인: echo $SHELL
  - 현재 수행 중인 셸 확인: ps

❑ 셸 변경

- 해당 셸의 이름을 입력
  - sh, csh, ksh, tcsh, zsh, …
  - 셸 프로그램이 설치되어 있지 않을 경우 실행되지 않는다.
- 빠져 나올 때는 exit

Bash

❑ Bash (Bourne Again Shell)

- GNU 표준 셸 → 리눅스 표준 셸
- 1988년 처음 배포
- 현재 버전 3.2

❑ Bash 정보

- 다운로드: http://www.gnu.org/software/bash
  - 리눅스에 bash가 설치되어 있지 않은 경우 또는 upgrade
- 매뉴얼
  - 온라인: http://www.gnu.org/software/bash/manual
  - 명령행에서 help 명령으로 도움말 기능 제공

명령 기본+

❑ More meta-characters

메타문자	의미	예
?	문자 하나	a? - ab, ac, a3, …
*	문자 여러 개	c*t - cat, chat, come at, …
[set]	set에 있는 하나의 문자	[abc] - abc 중 하나 [a-z] - 모든 소문자 중 하나 [-a-z] - -와 모든 소문자 중 하나
[!set]	set에 없는 하나의 문자	[!0-9] - 숫자가 아닌 문자 [0-9!] - 모든 숫자와 !
{ s1,s2,… }	s1 and s2 and …	b{ed,olt,ar}s - beds, bolts, bars ls *.{c,h,o} - 확장자가 .c, .h, .o인 모든 파일 리스트

명령 기본+

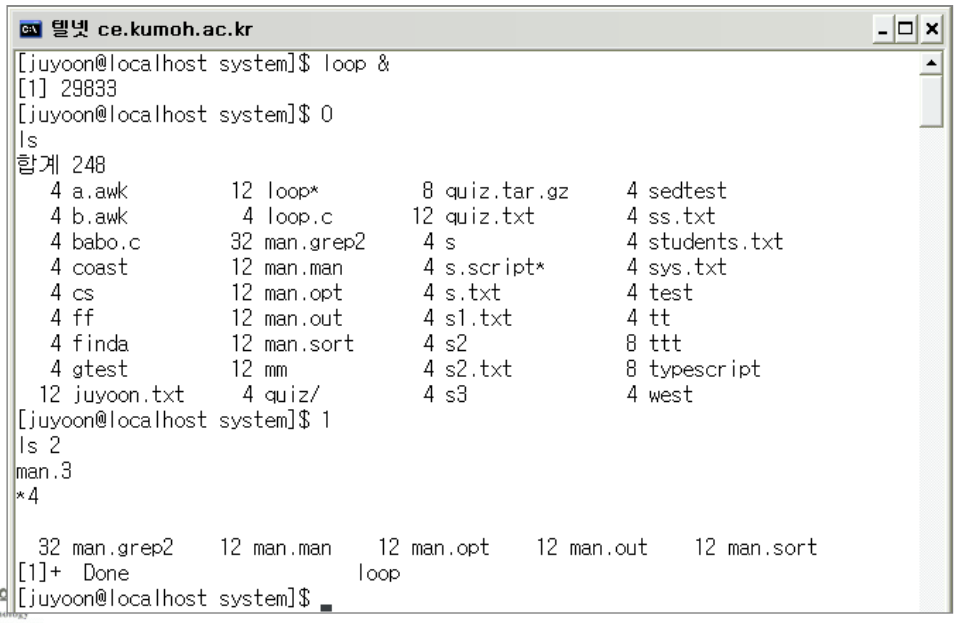
❑ 셸에서 의미를 가지는 특수 문자

메타문자	의미	메타문자	의미
~	홈 디렉터리	$	변수
`	명령 대체	&	백그라운드 작업
#	Comment	* ?	와일드카드
( )	하위 셸 시작/종료	\	문자 그대로
|	파이프	[ ]	문자 집합
{ }	명령 블록	;	셸 명령 분리
‘	강한 인용부호	“	약한 인용부호
<	입력 재지정	>	출력 재지정
/	경로명 분리	!	논리 NOT

명령 기본+

❑ 백그라운드 작업 &

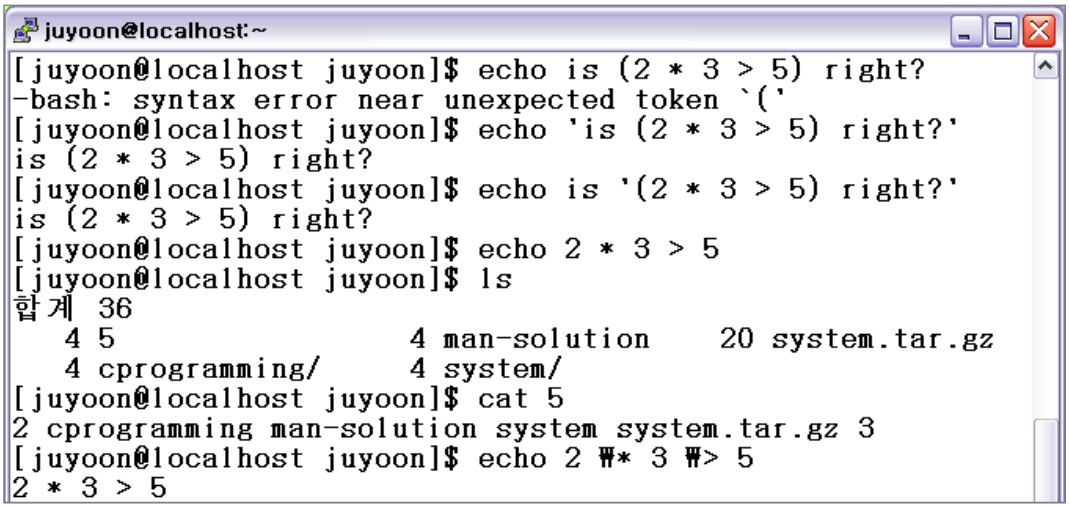
- 여러 작업을 동시에 수행
- 입력 없이 시간이 많이 걸리는 작업에 편리



명령 기본+

❑ 특수 문자를 일반 문자로 취급하려면?

- 인용부호 사용
- \ 사용



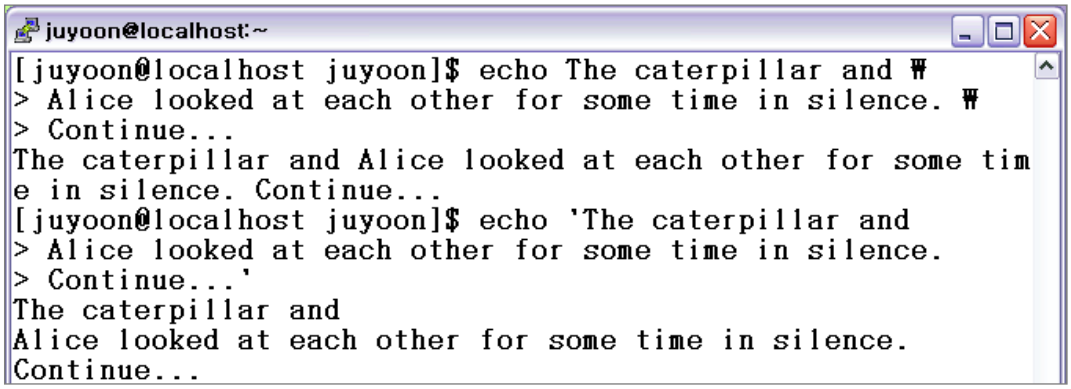
- 약한 인용부호 ": $, ?, \ 제외하고 일반 문자로 해석

Using Shell

명령 기본+

❑ 명령행의 계속

- '\n'을 일반 문자로 취급하기
- 행의 끝에 \ 사용
  - \n을 완전히 무시하고 한 줄로 연결
- 인용부호(')로 연결
  - \n을 명령의 끝이 아닌 하나의 문자로 취급



kit 국립금오공과대학교 Kumoh National Institute of Technology

명령 기본+

❑ 컨트롤 키

컨트롤키	stty 명	기능
^C	intr	현재 명령 중지
^D	eof	입력의 끝
^\	quit	^C가 동작하지 않을 경우 현재 명령 중지
^S	stop	화면 출력 정지
^Q	start	화면 출력 다시 시작
DEL, ^?	erase	마지막 문자 삭제
^U	kill	전체 명령행 삭제
^Z	susp	현재 명령 일시 중단

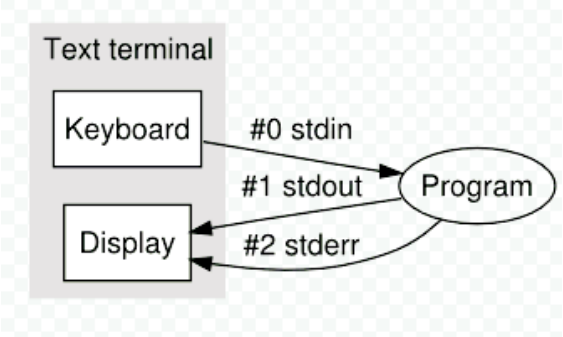
Redirection

❑ 표준 입출력

- 데이터가 유닉스 시스템에 저장되거나 전송되는 표준 방식
- 다양한 입출력 기기를 추상화하고 공통된 처리 방식을 사용하도록 최초로 시도

❑ 세 가지 표준 파일

- stdin (0) : 표준 입력
- stdout (1) : 표준 출력
- stderr (2) : 표준 오류 출력



http://en.wikipedia.org/wiki/Image:Stdstreams-notitle.svg

Redirection

❑ Redirection

- 쉘에서 제공하는 편의 기능
- 표준 입력 또는 출력을 파일 등 다른 입출력 장치로 보냄.
- 메타문자를 사용해서 redirection을 지시한다.

기 호	의 미
>	출력 redirection
>!	출력 redirection, csh의 noclobber 옵션을 중복 정의
>>	기존 파일에 출력을 추가
>>!	기존의 파일에 출력을 추가, csh의 noclobber 옵션을 중복 정의하고 파일이 존재하지 않으면 파일 생성
|	다른 명령으로 파이프 출력
<	입력 redirection
<<word	word로 시작하는 줄의 앞줄까지 표준 입력으로 받아들임
>&	표준 출력과 표준 에러를 파일로 redirection한다.
>>&	표준 출력과 표준 에러를 파일에 추가한다.

Redirection

❑ Redirection을 이용한 텍스트 파일 편집

```
juyoon@linda: ~/test/system
juyoon@linda:~/test/system$ cat > myfile
This is a test of redirection.
읽을 수 있어?
juyoon@linda:~/test/system$ cat myfile
This is a test of redirection.
읽을 수 있어?
juyoon@linda:~/test/system$
```

- 입력 끝(EOF) 표시는 ^d
- >>로 redirection: 기존 파일에 추가

```
juyoon@linda: ~/test/system
juyoon@linda:~/test/system$ cat >> myfile

이건 덧붙이는 거야. 끝낼 때는 ^D
juyoon@linda:~/test/system$ cat myfile
This is a test of redirection.
읽을 수 있어?

이건 덧붙이는 거야. 끝낼 때는
juyoon@linda:~/test/system$
```

Redirection

❑ 입력

```
juyoon@linda: ~/test/cprogramming
juyoon@linda:~/test/cprogramming$ getchar < getchar.c
#include <stdio.h>

int main(void)
{
        char ch=0;
        while (ch != EOF) {
                ch = getchar();
                putchar(ch);
        }
        return 0;
}
 juyoon@linda:~/test/cprogramming$
```

❑ 입출력 혼용

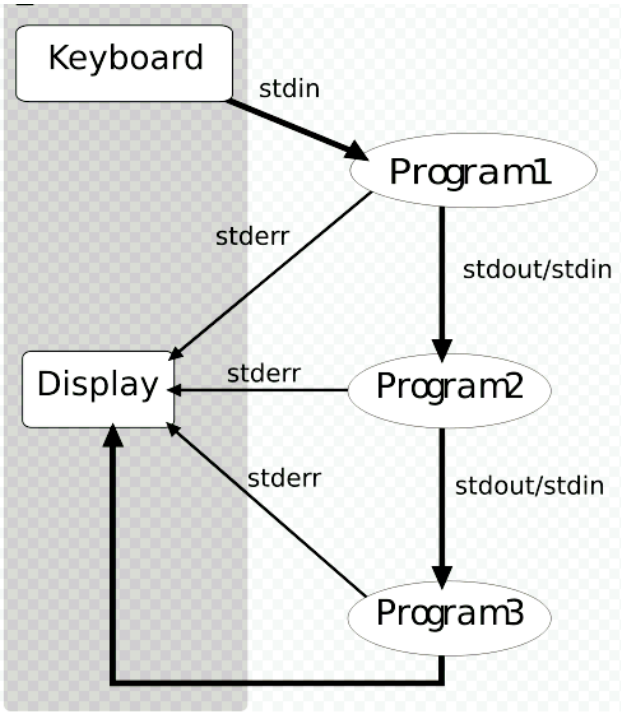
```
juyoon@linda: ~/test/cprogramming
juyoon@linda:~/test/cprogramming$ getchar < getchar.c > get.c
juyoon@linda:~/test/cprogramming$ cat get.c
#include <stdio.h>

int main(void)
{
        char ch=0;
        while (ch != EOF) {
                ch = getchar();
                putchar(ch);
        }
        return 0;
}
 juyoon@linda:~/test/cprogramming$
```


Redirection

❑ Pipe

- 한 프로세스의 표준 출력을 다른 프로세스의 표준 입력으로 사용
- 하나의 커다란 문제를 작은 작업으로 나누어 해결하는 것이 가능
- 예: man 명령의 결과를 파일에 저장

```
$ man 명령어 | colcrt > 파일
```




```
$ program1 | program2 | program3
```

http://en.wikipedia.org/wiki/Image:Pipeline.svg

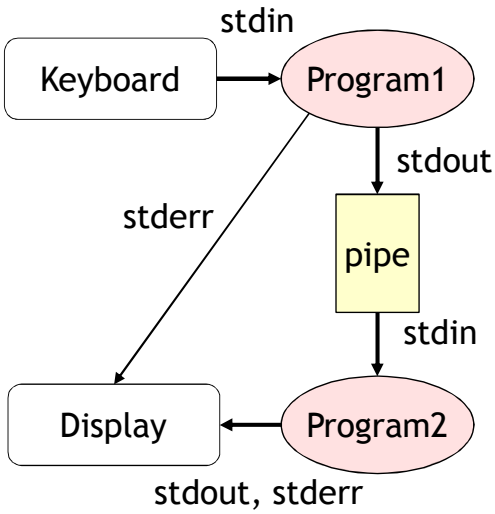
Redirection

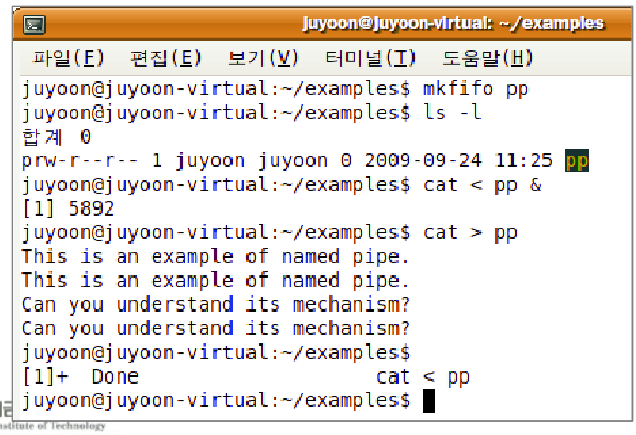
❑ Named Pipe

- 표준 입출력 외에 파일을 이용해 파이프 설정

```
$ mkfifo 파이프이름
```

- 사용은 redirection (<, >) 이용
- 두 개 이상의 프로세스가 함께 실행되어야 함.





명령행 편집

❑ 단순히 명령을 타이핑하고 Enter키 입력?

- 잘못 쓴 글자는? - Backspace
- 복잡한 명령을 재사용하고 싶으면?
- 이전에 했던 명령을 보고 싶으면?
- 긴 명령을 한꺼번에 혹은 한 단어씩 지우고 싶으면?
- 이전 명령을 조금만 바꿔서 다시 사용하고 싶으면?
- 기타 등등…

명령행
편집 기능!

명령행 편집

❑ 두 가지 모드

- Emacs 모드 / vi 모드
- vi 모드로 바꾸고 싶으면 'set -o vi'

❑ Emacs 모드 기본 명령

명령	기능	명령	기능
^B(←)	한 문자 왼쪽으로 이동	^F(→)	한 문자 오른쪽으로 이동
^D	오른쪽 한 문자 삭제	ESC-B	한 단어 왼쪽으로 이동
ESC-F	한 단어 오른쪽으로 이동	ESC-DEL	왼쪽 한 단어 삭제
ESC-D	오른쪽 한 단어 삭제	^Y	마지막 삭제 항목 되살림
^A	명령행 시작점으로 이동	^E	명령행 맨 끝으로 이동
^K	현재에서 끝까지 삭제		

명령행 편집

❑ 히스토리

- .bash_history에 사용한 명령을 기록
- 500개를 기록하도록 설정되어 있다.
- 'history': 히스토리 파일을 보는 명령

명령	기능	명령	기능
^P(↑)	이전 명령	!!	마지막 명령
^N(↓)	다음 명령	!n	n번째 명령
^R	뒤로 검색	!-n	n번째 앞의 명령
ESC-<	히스토리 파일의 첫 명령	!*string*	*string*으로 시작하는 마지막 명령
ESC->	히스토리 파일의 끝 명령	^*str1*^*str2*	마지막 명령의 *str1*을 *str2*로 바꾸어 반복

명령행 편집

❑ 내용 자동 완성

- TAB
  - 함수, 경로명, 파일명 등의 입력 시 텍스트 일부를 입력 후 TAB 키를 누른다.
  - 일치하는 것이 여러 개 있을 경우 가장 긴 것이 선택된다.
- 자동 완성 관련 키

명령	기능
ESC-?	선택할 수 있는 모든 경우 출력
ESC-/	파일명 자동 완성
ESC-~	사용자명 자동 완성
ESC-$	변수명 자동 완성
ESC-@	호스트명 자동 완성
ESC-!	명령 자동 완성
ESC-TAB	히스토리 목록에서 이전 명령의 자동 완성

명령행 편집

❑ vi 모드 명령행 편집

- vi 편집기 사용과 거의 같다.

❑ 나만의 편집 기능 정의

- emacs, vi 모드 다 싫다면?
- .inputrc 내에 readline 함수를 사용

.inputrc

```
Control-t: end-of-line
Control-o: "> output"
\e-b\e-x: backward-kill-word
...
```

- 자세한 내용은 매뉴얼을 참조하자.

사용자 환경 설정

❑ 시동 파일 (startup file)

- 셸이 시작할 때 여러 가지 환경 설정 등 기초 작업을 수행하는 명령들을 모은 파일
  - Bourne Shell(sh): .profile
  - C Shell(csh): .login, .cshrc
  - Korn Shell(ksh): .profile
- Bash: .bash_profile, .bashrc, .bash_logout
  - .bash_profile: 로그인 시 실행. 없으면 .bash_login, .profile 차례로 찾아 수행
  - .bashrc: 셸 수행 시마다 실행. 로그인 시에도 실행되도록 하려면 .bash_profile에 ". ~/.bashrc"를 포함시킨다.
  - .bash_logout: 로그아웃 시 실행.
  - 개별 시동파일 외에 /etc/profile, /etc/bashrc 등 실행

사용자 환경 설정

❑ 시동 파일 수정

- 환경 설정을 변경하고, 그를 로그인 시마다 변화 없이 유지하고 싶을 때
- vi 등의 편집기로 수정
- 효력이 생기려면?
  - 새로 로그인하거나 셸을 실행
  - source 또는 . 명령어를 사용하면 즉각 효력 발생

```
$ source .bashrc
```

❑ 그 외 환경 설정 파일

- .vimrc
- .emacs

사용자 환경 설정

❑ Alias

- 명령어의 별칭 또는 약어 정의

```
alias name=command
```

  - = 좌우에 공백 없어야 한다.
  - 순서에 따라 적용된다.
  - 현재 적용되는 모든 alias를 보려면?

```
$ alias
```

```
alias rm='rm -i'
alias ls='ls -sFC'
alias la='ls -a'
alias dir='ls'
alias cds='cd ~/system/quiz'
```

- 명령어 외의 alias?

```
$ alias ss=~/system/quiz
$ cd ss
```

  - 공백 문자로 끝나는 alias를 만들어 해결

- 고수가 되면? - alias보다 셸 스크립트/함수 활용

사용자 환경 설정

❑ 셸 옵션

- on 또는 off로 설정하여 동작을 제어

```
$ set -o optionname
```
옵션을 on

```
$ set +o optionname
```
옵션을 off

옵션	설명
emacs	emacs 모드 시작. 기본값은 on
ignoreeof	^D로 로그아웃하는 것 방지
noclobber	출력 재지정(>)시 overwrite 방지
noglob	와일드카드 확장 방지
nounset	정의하지 않은 변수 사용 시 오류 메시지
vi	vi 모드 시작

사용자 환경 설정

❑ 셸 변수 설정

- 셸 변수: bash 제공 내장 변수 + 사용자 지정 변수
  - on/off 만으로 설정하지 못하는 요소 설정
- 셸 변수 값 설정

```
varname=value
```

  - = 좌우로 공백이 없어야 하고 두 단어 이상은 ' '로 묶는다.
  - 다른 명령에 변수를 사용하려면 이름 앞에 $ 붙인다.
- 지정 변수 삭제

```
unset varname
```

- 변수값의 확인

```
echo $varname
```

- 설정한 변수값을 환경 변수로 효력 발생

```
export $varname
```


사용자 환경 설정

❑ 공통적인 내장 환경 변수

변수	의미	변수	의미
HOME	홈 디렉터리 경로	BASH_VERSION	실행 중인 bash의 버전
PATH	명령어를 탐색할 경로	PWD	현재 디렉터리
MAIL	메일박스의 절대 경로	HISTFILE	히스토리 저장 파일명
USER	사용자 ID	HISTFILESIZE	히스토리에 저장할 최대 명령 수. 기본값은 500
SHELL	로그인 셸의 절대 경로	EDITOR	기본 편집기 절대경로
TERM	터미널 유형	CDPATH	cd 명령 시 사용할 탐색 경로
PS1, PS2, PS3, PS4		프롬프트 문자열 변수	

사용자 환경 설정

❑ PATH 설정

- 명령을 입력했을 때 그 명령(의 실행 파일)이 어디 있는지 셸이 찾을 수 있게 해 주는 변수
- :로 구분하여 경로명을 입력한다.

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

- 이미 설정된 PATH에 덧붙일 때는 $PATH 활용

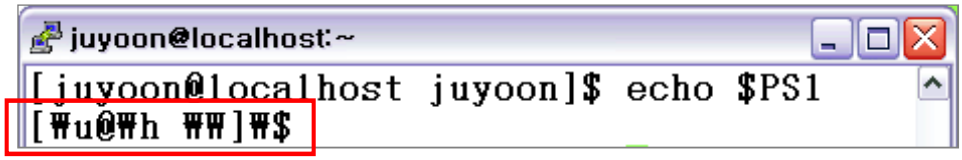
```
PATH=$PATH:/usr/local/bin
```

- 명령어를 입력하면 $PATH에 설정된 경로의 앞에서부터 찾는다. 즉, 같은 이름이 있으면 앞의 경로에 있는 것 실행
- 경로가 많으면 명령어 실행이 오래 걸릴까?
  - hash 명령을 한 번 실행해 보자.

사용자 환경 설정

❑ 프롬프트 설정

- 나만의 프롬프트를 만들자!
- PS1: 기본 프롬프트 문자열 변수



  - 미리 정의된 기호와 사용자가 선호하는 문자들을 사용해 만든다.
  - '\'가 해석되어야 하므로 약한 인용부호(")를 사용한다.

사용자 환경 설정

❑ 프롬프트 문자열 정의

기호	의미	기호	의미
\a	ASCII beep 문자 (\007)	\j	현재 수행 중인 작업의 수
\d	'요일, 월, 일' 형식의 날짜	\l	셸의 터미널 디바이스 명
\e	ASCII escape 문자 (\033)	\v	Bash의 버전
\H	호스트명	\V	Bash의 release
\h	처음 "."까지의 호스트명	\w	현재 작업 디렉터리(~/...)
\n	RETURN (carriage return + linefeed)	\W	현재 작업 디렉터리 (마지막)
\s	실행 중인 셸 이름	\!	현재 명령의 히스토리 번호
\T	HH:MM:SS 형식(12시간제)의 현재 시간	\#	현재 명령의 명령 번호
\t	HH:MM:SS 형식(24시간제)의 현재 시간	\$	UID가 0이면 #, 아니면 $ 출력
\@	am/pm 형식(12시간제)의 현재 시간	\nnn	8진수로 된 문자 코드
\A	HH:MM 형식(24시간제)의 현재 시간	\\	backslash
\u	현재 사용자명	\[\]	비출력용 문자열 시작과 끝

사용자 환경 설정

❑ vi 환경 설정

- .vimrc 파일에 자신만의 vi 환경을 설정
- 옵션, 키보드 매핑, 화면 구성(색깔 등), 프로그래밍 언어별 설정 등 가능

❑ 옵션

- 220개가 넘는 옵션

```
set option=value
```

  - vi 실행 시 ex-mode에서 직접 입력 가능 → 즉시 효력을 발생하나 다음 vi 실행 시에는 유지되지 않는다.
  - 옵션을 리셋: set nooption
  - 어떤 옵션의 값을 보고 싶으면: set option?
  - 어떤 옵션의 값을 디폴트 값으로 설정: set option&

사용자 환경 설정

❑ vi 환경 설정

- 옵션 예

```
set ic
set ai
set ts=4
set sm
set si
set bg=dark
```

ignorecase
autoindent
tabstop=4
showmatch
smartindent
background

- command도 미리 설정
  - syntax on
    - 언어의 문법에 따라 색깔을 달리 표현