

# C Shell

- /bin/csh
- 시작순서
  - ① \$HOME/.cshrc → 모든 shell
  - ② /etc/login → login shell 만
  - ③ \$HOME/.login → login shell 만
- .cshrc (run commands): 자주 쓰이는 별명을 설정함
- .login : \$TERM, \$PROMPT, \$PATH 등의 변수 값을 설정함

# 변수

- ① 지역 변수(단순 변수, 리스트 변수)
- ② 환경 변수

x 단순 변수의 값 할당

- `set {name [= word]}*`

(예) `% set` ⇒ 모든 지역 변수의 리스트가 나타남

`% set name = M.J. KIM`

`% set name = "M.J. KIM"`

`% echo $name`

# 변수

## x 단순 변수로의 접근

① \$name

⇒ 변수 name의 값 (독립 사용할 때)

② \${name}

⇒ 변수 name의 값 (접속 사용할 때)

③ \${?name} : 변수의 설정(존재) 여부 질의

⇒ 변수 name의 설정되어 있으면 1, 아니면 0의 값

# 변수

(예) % set verb = sing

% echo I like \$verbing

⇒ verbing: undefined variable

% echo I like \${verb}ing ⇒ I like singing

(예) % cat flag.csh

#

set flag ⇒ flag를 null 문자로 설정함

if (\${?flag}) then echo flag is set endif

% flag.csh ⇒ flag is set

# 변수

x 리스트 변수의 값 할당

- `set {name = ( {word}* ) }*`

(예)    `% set colors = ( red yellow green )`  
         `% echo $colors`  
             `→ red yellow green`

# 변수

## x 리스트 변수로의 접근

- ① `$name[selector]`  
⇒ 리스트 변수 `name`의 값 (독립 사용할 때)
- ② `${name[selector]}`  
⇒ 리스트 변수 `name`의 값 (접속 사용할 때)
- ③ `$#name`  
⇒ 리스트 변수 `name`의 원소들의 개수  
(독립 사용할 때)
- ④ `${#name}`  
⇒ 리스트 변수 `name`의 원소들의 개수  
(접속 사용할 때)

# 변수

cf. selector: start-end 형태,  
start가 명시 안되면 1로 간주, \*는 모든 범위

(예) % set colors = ( red yellow green )  
% echo \$colors[1] ⇒ red  
% echo \$colors[2-3] ⇒ yellow green  
% echo \$colors[4] ⇒ Subscript out of range  
% echo \$#colors ⇒ 3

# 변수

## x 리스트 변수의 추가

- 원래의 리스트에 원소를 더하고, 이들을 괄호로 묶어 원래로 치환

(예) %set colors = ( red yellow green )

%set colors[4] = pink

⇒ Subscript out of range

%set colors = ( \$colors blue )

%echo \$colors ⇒ red yellow green blue

# 변수

- C shell에 미리 정의된 여러 지역/환경 변수들

\$?0	\$<	\$argv	\$cdpath	\$cwd
\$echo	\$home	\$history	\$histchars	\$mail
\$noglob	\$notify	\$path	\$prompt	\$shell
\$status	\$time	\$verbose		
\$savehist		\$noclobber		
\$nonomatch		\$ignoreeof		

# 변수

(예) % cat var5.csh  
    echo -n "please enter your name: "  
    set name = \$<           # 입력줄을 취함  
    echo hi \$name, your current directory is \$cwd

# 변수

## x 환경 변수의 값 할당

- setenv name word

(예) % setenv TERM vt100

% echo \$TERM

## x 미리 정의된 환경 변수

- ① 공통적으로 미리 정의된 환경 변수

\$HOME \$PATH \$MAIL \$USER

\$SHELL \$TERM

- ② C shell 특유의 환경 변수

⇒ \$LOGNAME (셸 소유자의 사용자 id)

# 연산식

x C shell의 연산식

- ① 문자열 연산식
- ② 산술 연산식
- ③ 화일 지향적 연산식

# 변수

## x 문자열 연산식

연산자	의 미	연산자	의 미
==	같으면 참	==	==와 동일. 단, 오른쪽에 대표문자 포함
!=	같지 않으면 참	!=	!=와 동일. 단, 오른쪽에 대표문자 포함

# 변수

(예) % cat expr1.csh

#

echo -n "do you like C shell? "

set reply = \$<

if (\$reply == "yes") then

echo you entered yes

else if ( \$reply =~ y\* ) then

echo I assume you mean yes

endif

# 연산식

## x 산술 연산식 (C-like)

연산자	의 미	연산자	의 미
~	unary 음수	<= >= < >	관계 연산
!	논리적부정	== !=	같다, 같지 않다
* / %	곱셈, 나눗셈, 나머지	& ^	bit, and, xor, or
+ -	덧셈, 뺄셈	&&	logical and, or
<< >>	bitwise shift		

# 연산식

(예) % cat expr3.csh

#

set a = 3

set b = 5

if (\$a > 2 && \$b > 4) then

echo expression evaluation seems to work

endif

# 연산식

- 연산식의 결과를 변수에 할당하기 위해서는 set 명령을 사용하지 않는다.

@

⇒ 모든 셀 변수들을 표시하는 내장명령어

@variable op expression

⇒ 변수 variable에 expression의 결과 할당

@variable[index] op expression

⇒ variable의 index번째에 할당

# 연산식

(예) % set a = 2 \* 2      ⇒ set: Syntax error.

% @ a = 2 \* 2

% echo \$a      ⇒ 4

% @ a = \$a + \$a

% @ b = ( \$a && \$flag)

⇒ && 때문에 괄호 필요

% set value = 1

% @ value ++      ⇒ ++ 혹은 -- 연산 가능

# 연산식

## x 파일 지향적 연산식

option "fileName" ⇨ 만일 옵션이 참이면 1을, 아니면 0을 반환

만일 fileName이 존재하지 않으면 0을 반환

옵 션	의 미	옵 션	의 미
r	셀은 fileName에 대해 읽기 허가를 갖는다.	o	셀 프로그램과 fileName의 소유자가 동일하다.
w	셀은 fileName에 대해 쓰기 허가를 갖는다.	z	fileName이 존재하되 크기가 0이다.
x	셀은 fileName에 대해 실행 허가를 갖는다.	f	fileName은 정규화일이다.
e	fileName이 존재한다.	d	fileName은 디렉토리이다.

# 연산식

(예) % cat expr4.csh

```
#
```

```
echo -n "enter the name of the file you wish to erase: "
```

```
set filename = $<
```

```
if ( ! (-w "$filename") ) then
```

```
    echo you do not have permission to erase that file.
```

```
else
```

```
    rm $filename
```

```
    echo file erased
```

```
endif
```

# 파일 이름 완성

## 1. set filec

→ filec 변수를 먼저 설정한다.

## 2. 파일 이름의 일부를 입력하고 ESC 누름

→ 유일한 파일식별가능 : 나머지 자동추가

→ 식별불가능 : 삐 소리

일치되는 파일을 찾으려면 “\*” 사용

ex) \$ ls -al .log\*

# 별명(Aliases)

x 자신만의 고유한 명령어를 만들어 사용한다

- alias [word [string] ]

(예) % alias       ⇒ 현재의 별명들의 목록을 보여줌

(예) % alias word ⇒ word가 어떤 string의 별명인지를 보여줌

(예) % alias dir 'ls -aF'

      % alias ls 'ls -aF'

(예) % alias who 'date; who'

      ⇒ who에 대한 무한루프 발생(error)

      % alias who 'date; /bin/who'

      ⇒ who의 절대경로 사용 해결

# 별명 (Aliases)

x unalias pattern

- pattern과 일치하는 별명을 제거한다.
- 만일 pattern에 \*가 사용되면 모든 별명을 제거한다.

# 별명 (Aliases)

- x 유용한 별명 (.cshrc에 저장)

```
% alias cd 'cd W!*; set prompt="$cwdW!>"; ls
```

⇒ 지정된 디렉토리로 이동하고, prompt를 현재의 디렉토리와 마지막 명령 번호를 포함하도록 한 후, 현재의 목록을 보여줌

W!\* (두 번째부터 마지막번째까지의 토큰 즉 모든 인수들)

W! (마지막 명령어의 번호)

```
% alias ls 'ls -F'
```

⇒ 파일과 디렉토리에 관한 추가 정보 보이기

# 별명 (Aliases)

`% alias rm 'rm -i'`    ⇨ 지울지 말지 확인을 먼저 한다.

`% alias rm 'mv W!* ~/tomb'`

⇨ 지울 대상을 tomb 디렉토리 밑으로 옮김

`% alias h 'history'`    ⇨ 히스토리 정보 얻기

`% alias vi 'mesg n; /bin/vi W!*; mesg y'`

⇨ vi 편집 기간동안 타인으로부터의 메시지 전달 방해 억제

`% alias moer 'more'`    ⇨ 일어나기 쉬운 오타 방지

`% alias ls-l 'ls -l'`    ⇨ 일어나기 쉬운 오타 방지

`% alias ll 'ls -l'`    ⇨ 상세한 목록 정보 얻기

# 히스토리

- 키보드로부터 받은 명령어들을 순서대로 기억해둠
- ◆ 번호가 붙여진 명령어
  - 효과적인 사용을 위해, prompt에 명령어의 번호가 붙음

```
% set prompt = 'W! %'  
1 % echo hello
```

# 히스토리

## x 관련환경 변수

\$history

- 히스토리 목록의 크기, default = 1

\$savehist

- 히스토리 파일(\$HOME/.history) 에 저장되는 명령어의 수
- 세션 간에 명령어 접근을 허용하는 능력

cf. .history 파일은 같은 사용자에게 의하여 생성된 모든 대화형 C shell에 의하여 공유된다.

# 히스토리

(예) \$ % set history 40

⇒ 마지막 40개의 명령어를 기억하도록 함

(예) \$ % set savehist 32

⇒ 세션들 사이에 32개의 명령어를 저장함

# 히스토리

## x 히스토리 읽기

- history [-rh] [number]

옵션이 없으면 마지막 \$history 명령을 나열한다

-r                    히스토리의 역순으로 읽는다

-h                    사건 번호의 표시를 금지한다

number            읽을 명령어의 개수

(예) % history -r 3

⇒ 마지막 3개의 명령어를 역순으로 읽음

# 히스토리

## x 명령어의 재실행

- 재실행된 명령어 텍스트는 echo된 후 실행된다
  - ① !!                    마지막 명령의 텍스트로 치환된다
  - ② !number    명시된 사건 번호를 갖는 명령의 텍스트로 치환된다
  - ③ !prefix      prefix로 시작되는 마지막 명령의 텍스트로 치환된다
  - ④ !?Substring?   substring을 포함하는 마지막 명령의 텍스트로 치환(예) % !41    ⇨ 41번째 명령어를 실행함

# 히스토리

## x 히스토리 수정자(modifier)

- 사건 명시자 바로 뒤에 나와서 이전 명령의 일부에 접근

- |              |                               |
|--------------|-------------------------------|
| ① :0         | 첫 번째 토큰                       |
| ② :number    | (number+1)번째 토큰               |
| ③ :start-end | (start+1)부터 (end+1)번째 토큰      |
| ④ :^         | 첫 번째 토큰 (:은 생략 가능)            |
| ⑤ :\$        | 마지막번째 토큰 (:은 생략 가능)           |
| ⑥ :*         | 두 번째부터 마지막번째 토큰<br>(:은 생략 가능) |

# 히스토리

(예) 48 % echo I like horseback riding  
I like horseback riding  
49 % !!:0 !!:1 !!:2 !!:4  
echo I like riding  
I like riding  
50 % echo !48:1-\$  
echo I like horseback riding  
I like horseback riding

# 히스토리

## x 파일 수정자

- 기존의 히스토리 수정자에 덧붙여 파일 이름의 특정한 부분을 접근함
  - ① :h 파일명을 제외한 앞 부분
  - ② :r 파일의 루트(확장자제거) 부분
  - ③ :e 파일의 확장자 부분
  - ④ :t 파일의 뒤 부분 → 즉, 파일명만

# 히스토리

(예) 53 % ls /usr/include/stdio.h

⇒ /usr/include/stdio.h

54 % echo !53:1:h ⇒ /usr/include

55 % echo !53:1:r ⇒ /usr/include/stdio

56 % echo !53:1:e ⇒ h

57 % echo !53:1:t ⇒ stdio.h

# 히스토리

## x 대치 수정자

- 텍스트가 대치된 후 재실행됨

!event:s/pattern1/pattern2/

(예) 58 % ls /usr/include/stdio.h

⇒ /usr/include/stdio.h

58 % !58:0 !58:1:s/stdio/signal/

⇒ /usr/include/signal.h

# 제어구조

- C-like

x foreach-end

- 명령어 목록이 반복 실행됨, 반복시마다 해당 변수가 다른 값을 가짐

```
foreach  name ( wordlist )  
          commandlist  
end
```

# 제어구조

```
(예) % cat foreach.csh
      #
      foreach color (red yellow green blue)
          echo one color is $color
      end
% foreach.csh
one color is red
one color is yellow
one color is green
one color is blue
```

# 제어구조

x goto

- 무조건 분기



# 제어구조

x if-then-else-endif

if ( expr )      command
--------------------------

if ( expr1 ) then commandlist1 else if ( expr2 ) then commandlist2 else commandlist3 endif
--

# 제어구조

## x Onintr

- 키보드로부터 ^C (interrupt, SIGINT) 값을 받았을 때 분기하도록 지시

```
onintr [ - | label ]  
    - 인터럽트 무시  
    label label로 분기
```

# 제어구조

## x repeat

- 단일 명령어를 지정된 시간의 수만큼 반복 수행한다

```
repeat expr command
```

(예) % repeat 2 echo hi there   ⇒ 2개의 줄 표시

```
hi there
hi there
```

# 제어구조

## x switch-case-endsw

- 다중 분기 지원

```
switch (expr)
    case pattern1 :
        commandlist1
        breaksw
    case pattern2 :
    case pattern3 :
        commandlist2
        breaksw
    default :
        commandlist2
endsw
```

# 제어구조

## x while-end

- 참인 동안 명령어를 반복 수행함

```
while ( expr )  
    commandlist  
end
```

```
cf.    break  
        continue
```

# 개선점

x 명령어 재실행을 위한 최적 방법

- ^pattern1^pattern2

(예) % cc fil.txt → Can't open file fil.txt

% ^fil^file → cc file.txt

x 메타문자 { }

- a{b,c}d → abd acd(접두사, 접미사의 입력시간 감소)

(예) % cp /usr/include/{stdio,signal}.h . → 두 파일을 복사

# 개선점

(예) % echo \*p

prog1.c      prog2.c

% set noglob    →    대표문자 처리 금지

% echo \*p

\*p

(예) % echo \*a

echo: No match.

% set nonomatch

% echo \*a

\*a            →    에러 발생하지 않고 원래 패턴 인쇄

# 개선점

## x redirection

- ① 표준 에러 채널을 리다이렉션: `>&` 또는 `>>&`
- ② 예상외의 덮어 씌우기로부터 파일 보호: `$noclobber` 설정 (default: 비설정)

(예) `% (process1 > file1) >& file2`

→ 표준 출력은 file1으로 표준 에러는 file2로 저장함

(예) `% set noclobber`

`% cc a.c >& errors`

`errors: File exists.`

# 개선점

x 표준출력 뿐 아니라 표준에러도 pipe시킴

- |&

(예) % cc a.c |& more

(예)% (process1 > file1) |& process2

⇒ process1의 표준 출력은 file1으로 저장하고,  
명령어 그룹의 출력과 에러 채널을 process2로  
파이프 처리함

## 예제 : JUNK

`junk -lp {fileName}*`

지정된 파일을 지우는 `rm` 대신 사용하는 명령으로,  
파일을 지우지 않고 `$HOME/.junk`에 이동시킨다.

`-l` `$HOME/.junk`의 내용을 보여준다(list).

`-p` `$HOME/.junk`을 제거한다(purge).

만일 `.junk` 디렉토리가 없으면 자동으로 생성해준다.

```

#!/bin/csh
# junk script
# 저자 그래엄빌
#
# 변수의 초기화
#
set fileList = ( )      # 모든 명시된 파일의 목록
set listFlag = 0        # “-l” 옵션이 명시되면 1로 설정
set purgeFlag = 0      # “-p” 옵션이 사용되면 1
set fileFlag = 0        # 적어도 1 파일이 명시되면 1
set junk = ~/.junk      # junk 디렉토리
#
# 명령줄 파싱
#
foreach arg ($*)
  switch ($arg)
    case “-p”:
      set purgeFlag = 1
      breaksw
    case “-l”:
      set listFlag = 1
      breaksw
    case -*:
      echo $arg is an illegal option
      goto error
      breaksw

```

```

default:
  setFlag = 1;
  set fileList = ($fileList $arg) #리스트에 추가
  breaksw
endsw
end

#
# 너무옵션이 많은가 조사
#
@total = $listFlag + $purgeFlag + $fileFlag
if ($total != 1) goto error
#
# junk 디렉토리가 없으면 이를 생성
#
if( !(-e $junk)) then
  ‘mkdir’ $junk
endif
#
# 옵션처리
#
if ($listFlag) then
  ‘ls’ -lgF $junk      #junk 디렉토리 나열
  exit 0
endif
#

```

```
#
if ($purgeFlag) then
    'rm' $junk/*          #junk 디렉토리의내용제거
    exit 0
endif
#
if ($fileFlag) then
    'mv' $fileList $junk  #파일을 junk 디렉토리로 이동
    exit 0
endif
#
exit 0
#
# 에러 메시지 표시 후에 종료
#
error:
cat << ENDOFTEXT
Dear $USER, the usage of junk is as follows:
junk -p means "purge all files"
junk -l means "list junked files"
junk <list of files> to junk them
ENDOFTEXT
exit 1
```

# 실습 과제

- Bourn Shell 에서 실습했던 5개 문제들을 C Shell 로 만들어 보세요
- C Shell 로 만든 junk 를 bourn Shell 로 만들어 보세요