Linux Shell

2008. March

KITRI - 한국정보기술연구원

Gin Valentine

VI Editor

- 오랜역사와 다양한 기능을 가진 편집기

  현재 vi 의 개선된 버전인 VIM(VI Improved)를 사용

- vi --help 다양한 실행옵션

  넓은 화면을 사용하기 위해 메뉴가 없으며 숨겨진 기능들을 포함 유닉스에서 필수적인 편집기

VI Editor (Cont')

실행 방법: vi 명령어 뒤에 파일 이름

vi 실행시 시작하는 모드는 명령 모드

입력모드 혹은 편집모드 – 글자를 입력할 수 있는 모드

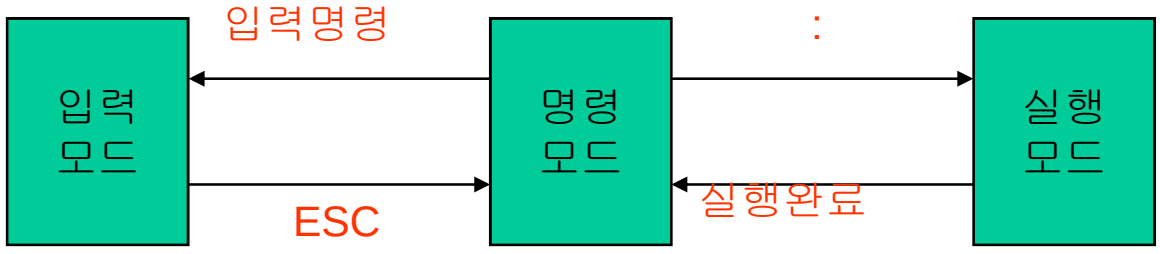
- 모드변환방법 – 명령모드에서 a,A,i,o,O를 입력 했을때

명령모드 혹은 ESC모드 – 커서이동 및 기타 명령어처리

- 모드변환방법 – 실행모드 혹은 입력모드에서 ESC키를 눌렀을 때

실행모드 혹은 콜론모드 – 내용바꾸기 및 기타

- 모드변환방법 – 명령모드에서 콜론(:)을 입력했을때



명령 모드: 커서 이동

한문자씩 이동
- h : 왼쪽, j : 위로, k : 아래로, l : 오른쪽
- 방향키 사용 가능

단어단위 이동
- w : 다음단어로, b : 이전단어로

행단위 이동
- ^ : 맨왼쪽의 첫글자, $: 마지막글자의 끝

화면 이동
- ^F: 한화면 아래로, ^B: 한 화면 위로, ^D: 반 화면 아래로, ^U: 반화면 위로

명령 모드: 입력 모드 전환

a : 커서 위치의 다음 칸부터 입력하기(append)

A : 커서가 있는 줄의 끝부터 입력하기

i : 커서 위치부터 입력하기 (키보드의 Insert도 같은 기능을 합니다.)

I : 커서가 있는 줄의 맨 앞에서부터 입력하기

o : 커서 바로 아래에 줄을 만들고 입력하기(open line)

O : 커서 바로 위에 줄을 만들고 입력하기

s : 커서가 있는 단어를 지우고 입력하기

S : 커서가 있는 행을 지우고 입력하기

명령 모드: 삭제

x : 커서 위치의 글자 삭제

X : 커서 바로 앞의 글자 삭제

dw : 한 단어를 삭제

d0 : 커서 위치부터 줄의 처음까지 삭제

D : d$ 커서 위치부터 줄의 끝까지 삭제

dd : 커서가 있는 줄을 삭제

dj : 커서가 있는 줄과 그 다음 줄을 삭제

dk : 커서가 있는 줄과 그 앞줄을 삭제

명령 모드: 복사 및 붙여 넣기

yw : 커서 위치부터 단어의 끝까지 복사하기
y0 : 커서 위치부터 줄의 처음까지 복사하기
y$: 커서 위치부터 줄의 끝까지 복사하기
yy : 커서가 있는 줄을 복사하기
yj : 커서가 있는 줄과 그 다음 줄을 복사하기
yk : 커서가 있는 줄과 그 앞줄을 복사하기
p : 커서의 다음 위치에 붙여 넣기
P : 커서가 있는 위치에 붙여 넣기

* 삭제된 것은 언제나 복사에 되어있음.

명령 모드: 그외 유용한 명령

작업 취소

- u : 작업 취소하기 (undo)
- U : 그 줄에 행해진 작업 모두 취소하기

반복:

- . : 조금 전에 했던 명령을 반복하기

대소문자 변환

- ~ : 대소문자 전환

검색

- /검색어 : 아래방향으로 찾기
- ?검색어 : 위방향으로 찾기
- n : 다음찾기

실행 모드

치환관련 실행

- :s/old/new/g
  - old를 new 로 치환
- :s/^old/new/g
  - 행의 첫단어가 old 인것을 new 로 치환
- :s/old$/new/g
  - 행의 끝단어가 old 인것을 new 로 치환
- :s/aaa//g
  - aaa를 삭제

파일 관련 실행

- :w 파일명 “파일명”으로 저장
- :q 저장하지 않고 종료
- :q! 변경 사항을 버리고 종료
- :e 파일명 “파일명”의 파일을 불러들여 편집
- :r 파일명 “파일명”의 파일을 읽어서 삽입
- :!명령어 외부명령어 실행

Linux Shell

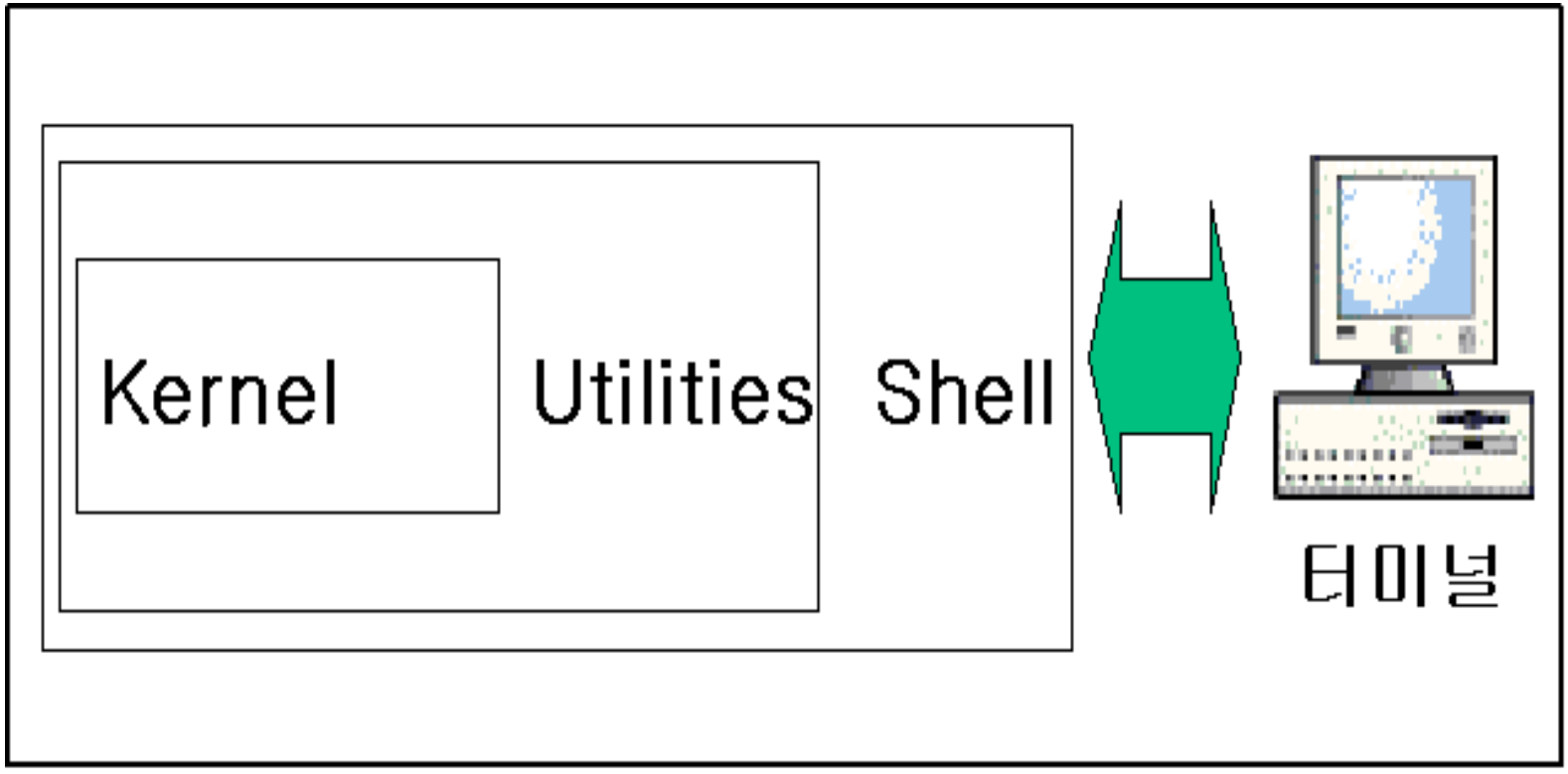
- 사용자가 입력한 명령어를 읽어서 해석하는 프로그램으로 명령어 해석기(Command Processor)라고도 함
- 쉘은 유틸리티(Utility)와 커널(Kernel)사이에 위치해서 상호작(Interface)을 담당
- 사용자가 명령어를 입력하면 셸은 특수한 작업을 수행하기 위하여 요구된 기계어의 집합(System Call)으로 명령어를 변환하여 실행
- 실제로 우리가 사용하고 있는 명령어는 이와 같이 명시된 시스템 콜(System Call)의 집합을 사용자가 사용하기 쉽고, 기억하기 쉽도록 만들어 놓은 것
- 쉘은 인식할 수 있는 "스크립트(Script)" 파일을 만들어서 리눅스의 일반적인 명령어뿐만 아니라 다음에 설명할 특별한 셸 프로그래밍 언어도 포함시킬 수 있음

Why Shell

- 새로운 시스템을 사용해야 하는 경우 GUI 환경은 다를 수 있으나 쉘 환경은 같음
- 시스템이 비정상적으로 동작하는 경우에 GUI는 지원되지 않을 수 있음
- 작업 속도
- GUI가 정확하게 원하는 동작을 제공하지 않을 경우도 있음
- GUI는 다양한 error 메시지나 상황을 전달하지 못함
- GUI는 script를 지원하지 않음

Shell Structure

리눅스 시스템의 구성요소



Sort of Shell

쉘의 종류

쉘의 종류	bsh	bash	csh	ksh	zsh	tcsh
개발자	Steven Bourne	Brain fox chet Ramey	Bill joy	David Korn	Paul Falstad	Ken greer
개발 연도	1979	1989	1981	1986	1990	1982
프롬프트	$	#	%	$	%	>

Bash 쉘 초기화 파일들

로그인을 통하여 bash를 사용할 때 다음과 같은 초기화 파일들을 사용함

/etc/profile

- 시스템 수준의 bash 초기화 파일
- PATH, USER MAIL, HOSTNAME, HISTSIZE 등과 같은 기본적인 쉘 변수 값을 설정함
- Java, Database와 같은 시스템에서 제공하는 기본 프로그램 사용환경도 설정함

/etc/bashrc

- Bash 고유의 설정 파일

~/.bash_profile

- Bash 개인별 설정 파일

~/.bash_login

- Bash 개인별 로그인 쉘에서 설정 파일

~/.profile

- ~/.bash_profile과 ~/.bash_login이 없을 경우 사용하는 파일

~/.bashrc

- Bash에서 사용하는 alias나 함수를 설정하는 파일
- 일반적으로 ~/.bash_profile에서 호출됨

사용자 환경파일 다루기

- 사용자가 처음 로그인하면 사용자 홈 디렉토리에 있는 ".bashrc"이라는 셸이 실행
- 이 파일 안에는 사용자의 터미널 타입, 홈 디렉토리, 검색 경로, 셸 정보, 시간설정 등 매우 다양한 사용자 환경을 지원하는 설정 값들이 들어 있음
- 사용자는 이러한 ".bashrc"를 수정하여 자기 나름대로의 취향에 맞게 수정하여 사용할 수 있음.
- ".bashrc"은 일반 텍스트 파일이기 때문에 편집기로 수정할 수 있음. 또한 이 파일은 "." 파일이기 때문에 일반 "ls" 명령으로는 보이지 않으므로 "-al" 옵션을 사용하여야 볼 수가 있음

Shell Change

- 리눅스에서 제공하는 셸의 종류를 알아보기 위해서는 "/bin" 디렉토리나 “/etc/shells" 파일을 확인
- 리눅스에서 제공하는 셸은 크게 로그인 셸(login shell) 과 서브셸(sub shell) 나눌 수 있으며, 로그인 셸은 사용자를 등록하면 자동적으로 실행되게 하는 셸을 말하며, 서브셸은 사용자가 필요에 따라 그때그때 임시적으로 변경하여 사용하는 셸을 말함
- 특정 셸로 변경하기 위해서는 단지 해달 셸 프로그램을 실행시키면 되며 종료하기 위해서는 단지 “exit"명령만 입력
- 로그인 셸을 변경하기 위해서는 “chsh(change shell)" 명령을 사용

정보 표시(Echo)

화면에 특정 정보(문자)를 표시해 줌

리눅스에서는 주어지는 인수(Parameter)를 표준출력에 표시하는 명령어로 "echo"를 사용

파라메터

파라메터	기능
\a	경고문자
\b	백 스페이스
\c	새로운 라인 없이 프린터라인 사용
\f	폼 피드(Form Feed)
\n	새로운 라인
\r	캐리지(Carriage Return)
\t	탭(Tap)

메타(Meta Character) 문자

쉘은 어떤 문자들을 아주 특수하게 인식하는데 이러한 특수 문자를 메타 문자라고 함

보통 쉘은 어떤 문장을 해석할 때 이러한 메타문자가 있는지를 확인하고, 만약에 존재하면 이러한 메타문자에 대한 특별한 기능을 수행

이때 주의해야 할 것은 문장에 메타문자가 들어가면 이 의미를 없애야 하는데 이때 사용하는 문자가 바로 "\“임

메타문자 들

| (bar), & (ampersand), ; (semi-colon),
() (parentheses), < > (redirect)

표준 입,출력 제어(Redirection)

입출력 제어

기호	의미
>	표준출력을 파일에 기록
>>	표준출력을 파일의 끝에 덧붙임
<	파일로부터 입력을 읽어 들임

">"와 ">>"은 매우 동일한데 ">>"은 기존 파일이 존재하면 맨 뒤에 덧붙이고, 그렇지 않으면 ">"로 동일한 기능을 수행

">"를 사용할 때 주의해야 할 것은 출력파일이 이미 존재하면 기존파일을 덮어써 버림. 그래서 ">"나">>"를 쓸 때에는 항상 기존 파일이 있는가를 확인해야 함

"<<"를 사용할 수 있는데, 이것은 특정한 파일이름으로 시작하는 줄의 앞줄까지의 표준 입력을 임시 파일로 복사하고, 그 임시 파일을 다시 표준 입력으로 사용하여 문장을 실행

와일드 문자 사용

특정한 패턴을 만족하는 파일들을 선택해 줄 수 있게 하는 대표(wild)문자를 제공

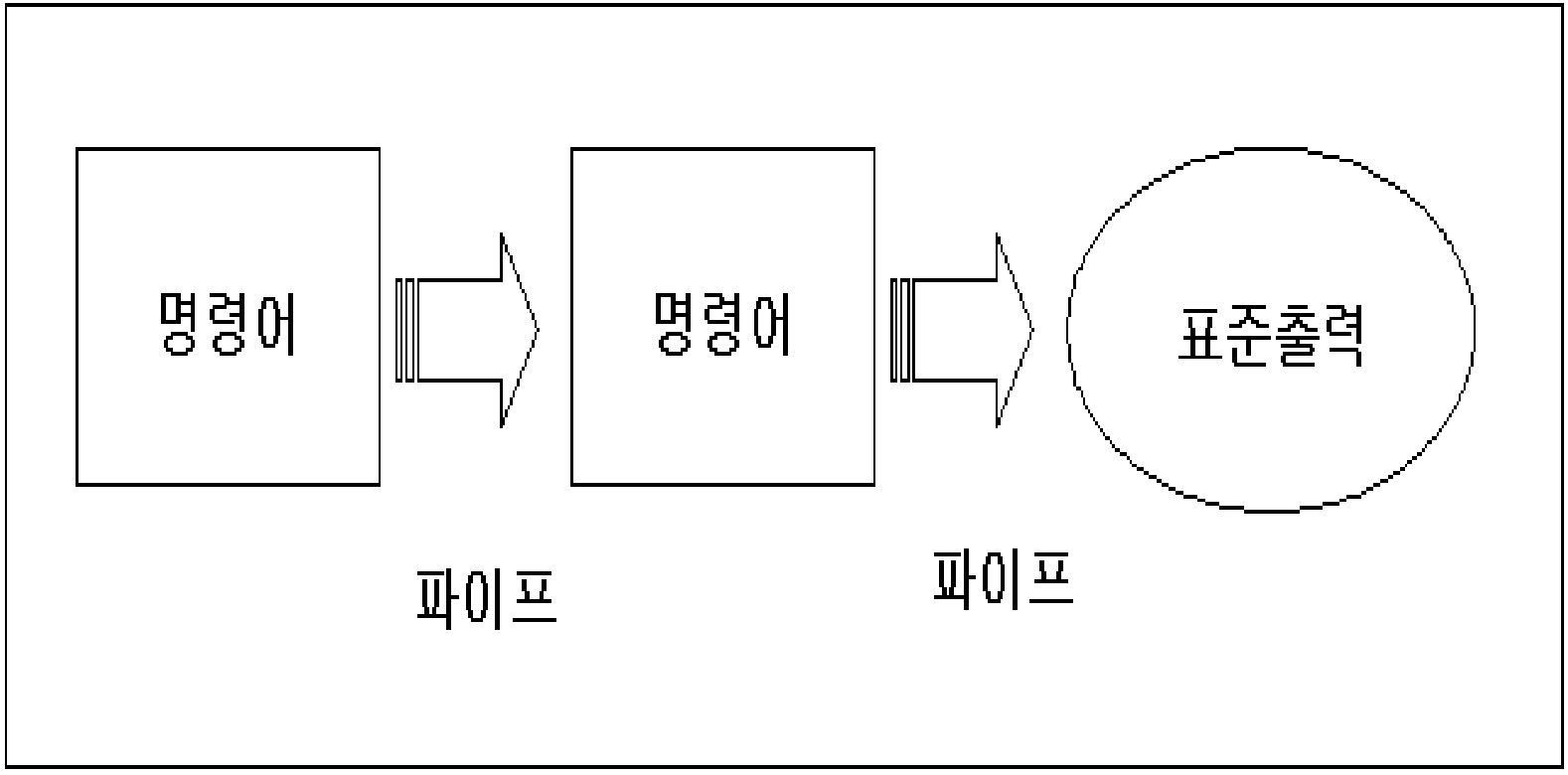
대표문자	의미
*	모든 문자열
?	한 문자와 일치
[..]	적어도 []안에 하나의 문자와 일치, "-"를 이용하여 범위지정

"/“를 사용할 때에는 경로 또는 디렉토리 인지를 확인한 후 사용해야 함

파이프 파인(Pipe Line) 사용

- 파이프는 바로 한 개의 명령 또는 그 이상으로 구성된 셸을 다른 명령어나 셸의 입력으로 사용할 수 있도록 하는 것을 말함
- 입력과 출력은 계속 연속적으로 연결될 수 있는데, 이것을 파이프라인(Pipe Line)이라고 함
- 파이프라인은 큰 작업이나 큰 시간이 걸리는 작업을 빠르고 단순하게 할 수 있는 장점 때문에 계속 발전
- 중간단계의 파이프라인 값을 저장하는 기능을 하는 "tee"라는 명령어를 사용
- "tee" 유틸리티는 2개의 파라미터를 갖는데 "-a" 옵션은 입력을 파일에 덮어 씌우지 않고 대신에 그 파일의 끝에 덧붙임. 이에 반해 "-i" 옵션은 인터럽트를 무시하도록 함.

명령어의 파이프 라인 동작 상태



명령어 관리(Command)

명령어의 처리 중에는 악센트 부호인 "`"를 사용할 수 있는데 만약에 "`"로 둘러싸인 부분은 일반문장으로 해석하지 않고 명령어로 해석하여 그 결과를 표준출력으로 출력함

간단한 명령어를 사용할 수 있지만 보다 복잡한 명령어들로 구성된 파이프라인을 사용할 수 있음

특정 정보를 포함하고 있는 소스파일, 셸, 특정 루틴을 검색하고 편집할 때 매우 유용하게 사용할 수 있음

명령어들의 실행순서(Sequences)을 한 행에 ";"로 구분할 수 있음, 파이프라인과 마찬가지로 실행순서는 왼쪽에서 오른쪽으로 진행

각 ";" 사이에는 파이프라인, 리다이렉션도 가능

조건부 실행

조건부 실행

문자	의미
&&	이전명령이 정상종료인 "0"의 값을 반환 할 경우에만 다음명령 실행
||	이전명령이 비정상종료인 "1"의 값을 반활 할 경우에만 다음명령실행

괄호로 묶어서 하나의 셸처럼 할 수도 있음, 이때에는 표준 입력, 표준출력, 표준 입,출력을 공유하며 하나의 명령어처럼 사용할 수 있으며, 리다이렉션이나 파이프라인 처리도 가능

Background Processing

- 작업의 처리는 크게 두 가지로 나눌 수 있는데, 포 그라운드(Foregr ound) 처리와 백그라운드(Backgr ound)처리로 나눌 수 있음
- 사용자가 어떤 작업의 결과를 중간에 볼 필요가 없고 오직 결과값 만을 원할 때, 또는 시간이 많이 걸리는 작업은 백그라운드로 작업을 하여 프로세서의 사용을 극대화 할 수 있음
- 백그라운드 작업은 해당 명령어나 셸에 "&"를 붙여주면 됨
- 백그라운드 작업이 시작되면 시스템은 현재 작업의 제어를 백그라운드에 프로세서를 별도로 할당하고 , 그 작업번호인 프로세서 ID를 사용자에게 보여주고 바로 명령어를 수행할 수 있는 상태로 변경
- fg 와 bg 그리고 jobs 명령어를 이용하여 작업을 포 그라운드와 백 그라운드간 이동을 할 수 있음

환경변수 목록

> 사용자가 시스템을 사용하기 위해 필요한 각종 설정 값들을 정해주는 역할을 함

> 여기에서 변수란 값이 수시로 바뀔 수 있다는 것을 뜻하며, 변수의 종류에는 사용자가 설정할 수 있는 변수가 있는 반면 문자열로서 특정한 값을 저장할 수 있는 변수가 있음

> 변수들은 필요할 때마다 셸 사용자에 의해 만들어질 수 있으며, 스크립트를 작성할 때 매우 유용

> 환경변수 설정 값들을 보기 위해서는 공통 명령어인 “set”를 사용

중요 환경 변수들

환경변수들의 용도

환경변수	의미
HOME	사용자가 처음 로그인 할 때 홈 디렉토리 위치를 저장
PATH	셸이 사용자가 입력한 명령이나 파일을 찾을 때 검색하는 경로
PS1,PS2	셸이 사용하는 기본 프롬프트와 두 번째 프롬프트
MAIL	사용자의 편지가 저장된 파일의 이름을 명시
MAILCHECK	셸 편지가 도착하였는지를 검사하는 시간을 설정
MAILPATH	파일명의 목록을 포함
SHELL	사용자 환경에서 현재 사용하고 있는 셸을 명시
TERM	터미널 종류를 설정
TERMCAP	터미널 종류에 대한 정보를 포함
TERMINFO	터미널 종류에 대한 정보를 포함하고 있는 디렉토리를 명시
TZ	시간대 (Time Zone)을 정의

환경변수 표시 및 설정

"set" 명령어는 현재의 환경변수의 목록을 나타내 줌

사용자는 이러한 환경변수를 표시하고 수정할 수 있게 함

셸 변수를 다시 정의함으로써 시스템 프롬프트나 탐색경로를 변경시킬 수 있음

어떤 명령어의 출력으로 셸 변수를 정의하고 사용할 수 있음

간단한 명령에서부터 파이프라인과 같은 좀더 복잡한 명령어 식을 넣을 수도 있음

인용부호 사용(Quoting)

각종 셸의 대표문자를 금지할 필요가 있을 때가 있다. 이러한 경우에는 보통 인용부호인 "", " 를 사용

인용부호 종류

인용부호	의미
''	대표문자의 대치와 변수대치($), 명령어 대치 금지(`)
" "	대표문자의 대치만을 금지
중첩	인용부호가 중첩될 때에는 바깥쪽의 인용부호만 효력을 가짐

별명(Alias)

> 별명은 글자 그대로 명령어의 또 다른 별칭

> 보통 자주 사용하는 명령이나 자신의 취향, 옵션이 반드시 필요한 경우 등에 자주 사용

> 별명으로 사용할 워드(word)와 실행명령을 콜론(')으로 감싸서 "="으로 연결하면 첫 번째 워드가 실행명령으로 대치되고 그 명령이 실행

> 만약 "alias" 명령만 입력하면 현재 저장되어 있는 모든 별명의 목록이 표시된다. 만약 이미 별명이 존재하면 새로운 별명으로 대치 됨

> 별명을 이용하여 다른 별명을 만들 수도 있음

> Shell에서 제공하는 예약어(Reserved word), 즉 내장 명령어를 사용하면 원래의 의미를 상실

> 정의한 별명을 제거하기 위해서는 "unalias" 명령을 사용

별명(Alias) Cont'

미리 정의된 주요 별명

별명	값
autoload	'typeset -fu'
command	'command '
functions	'typeset -f'
history	'fc -l'
integer	'typeset -i'
local	typeset
nohup	'nohup '
stop	'kill -STOP'
suspend	'kill -STOP $$'

히스토리(History) 기능

Bourne 셸은 사용자가 입력한 명령을 순서대로 특정파일에 저장

히스토리 기능은 지금까지 사용한 명령어를 다시 불려내어 편집이 가능하도록 함

히스토리 기능은 이전에 수행된 명령어를 다시 실행할 수 있도록 하는 단순기능과 이전 명령을 불러와서 앞장에서 배웠던 에디터처럼 편집하여 실행시킬 수 있는 방법을 제공

히스토리를 사용할 때, 프롬프트에 입력하려는 명령어의 "번호"를 포함시킬 수 있는데 이것은 나중에 정렬할 수 있게 하기 위해서 임

히스토리 목록을 관리하기 위하여 두 개의 환경변수 "$HISTSIZE" 와 "$HISTFILE"을 사용

이전의 명령어들을 재실행 할 수 있게 하는데, 이때 사용하는 것이 바로 "fc" 명령어

종료상태(Exit Status)

> 셸 명령어들은 실행이 끝났을 때 현재 그 상태를 반환

이 상태는 그 명령이 성공하였는지, 아니면 실패하였는지를 셸에게 알려줌

> "$?"은 가장 최근에 실행된 명령의 성공여부를 보여줌

> "exit"명령은 셸이 수행도중에 실행을 끝나고 그 결과를 반환할 수 있음. 셸이 스크립트 안에서 "exit"명령을 만나면 해당 스크립트 수행이 끝나게 됨

> 보통 "exit" 명령은 뒤에 숫자를 붙어 종료 상태를 나타내게 됨

Shell Execution

- 쉘 실행방법은 강제로 쉘을 실행시키는 방법과 퍼미션을 조정하여 실행시키는 방법이 있으며, 강제로 셸을 실행시키는 명령어는 "bash“임
- 일련의 명령어들을 필요에 따라, 혹은 나중에 실행하기 위해 텍스트형태로 저장할 수 있고 이를 실행할 수 있는데 이 파일을 스크립트(Script)라고 함
- 번역기가 각 문장을 나씩 차례대로 읽어서 실행하기 때문에 일반적으로 컴파일 된 파일보도 실행속도가 늦다. 그러나 이러한 스크립트가 사용되는 이유는 어떤 운영체제에서나 쉽게 이식할 수 있으며 특별한 과정이 필요 없이 실행할 수 있기 때문에 많이 사용.
- 대표적인 스크립트 언어로는 Java, Perl, Lisp, Tcl 등이 있음

좀더 개선된 기능

이전 셸에서는 변수의 사용이 9개까지만 사용이 가능해 졌지만 BASH 에서는 무제한으로 사용할 수 있음. 만약 파라메터가 11번째라면 “{}”를 사용하여 “${11}” 으로 사용

파라메터인 "$#", "$@", "$*"를 사용할 수 있음. "$#"은 현재 파라메터의 총 개수를 표시해 주며, "$@"와 "$*"는 매우 비슷하게 동작하나 "$@"는 각 파라메터를 독립된 인수로 취급

"select" 제어구조 명령은 간단한 메뉴를 쉽게 구성해 주고, 사용자의 선택에 대응하게 해 줌

BASH에서 사용하는 함수는 좀더 확장된 함수 기능으로, 보통 메모리에 존재하므로 실행 속도가 빠르며, 모듈화를 제공하여 셸을 일반화 시켜주고 더 나은 구성을 제공해 주고 있음

매개변수 (Parameter)

쉘 변수의 한가지 다른 타입은 쉘이 수행될 때 매개변수로 전달하는 것

보통 위치 매개변수(Positional Parameter)라고 하며 접근할 때 숫자를 사용

내장 변수

이름	의미
$$	셀의 프로세스 ID
$0	셸 스크립트의 이름
$n	명령어 줄의 n번째 인수를 나타냄
$*	모든 명령어 줄 인수의 목록

BASH 작성 및 실행

Bourne 셸은 Stephen bourne에 의해 만들어진 최초의 대중화된 Unix 셸로써 대부분의 Unix 셸에서 작동

리눅스의 기본 셸이기도 한 Bourne again 셸은 bourne셸의 기능을 포함하고 있으며 보다 진보된 기능을 제공

이 Bourne셸의 모든 기능을 포함하고, 다음에 설명할 다른 셸들의 기초를 제공하며, 아주 다양한 프로그래밍 언어를 지원

셸 프로그래밍은 기본적으로 셸 스크립트(script)은 앞에서 배운 "ed", "vi" 편집기를 사용. 셸을 실행하는 방법에는 리다렉션, 실행 명령어로 만들어 실행하는 방법, "sh" 이나 “bash" 명령어를 사용해서 실행하는 방법이 있음

입력 리다이렉션(<)을 이용하는 방법은 잘 사용하지 않음, 단점은 스크립트에 매개변수를 제공할 수 없다는 것임

BASH를 이용하는 방법은 실행모드를 확인할 필요가 없고 간단하므로 자주 사용됨

스크립트 방법은 실행모드를 확인하고 실행권한을 주는 불편함이 있으나 하나의 명령어로 사용하기 때문에 나중에 명령어로 사용하기 위해서는 대부분 이 방법을 사용

User Define Variable

사용자 변수에 값을 할당하려면 "=" 기호를 사용하며, 사용자는 이러한 변수 명은 문자나 첫 글자가 문자인 숫자의 조합으로 만들 수 있음

만약 할당하려는 변수가 존재하지 않으면 자동적으로 새로운 변수를 생성하지만, 만약 기존 변수가 존재하면 이 변수가 가지고 있는 이전 값을 새로운 값으로 변경

이때 주의해야 할 것은 값을 할당할 때 "=" 사이에 공백을 주어서는 안 됨

변수에 할당되어 있는 값을 출력하려면 지금까지 가장 많이 사용한 "echo"명령을 사용. 사용자가 변수를 참조할 때는 언제나 변수 명 앞에 "$"를 사용

변수 값을 표시하기 위하여 "$"자체를 표시할 경우도 있는데, 이때에는 "$"앞에 "₩"를 넣어주거나 인용부호 ("")로 묶어줌

문자열과 문자열 상에서 많은 공백이나 탭을 포함하고 있는 문자를 출력할 때에는 변수 명 주위에 인용부호로 또한 묶어 줌

특수문자 "*"나 "*", "." 등이 포함되어 있어도 인용부호로 묶어 줌

할당되어 있는 변수값을 삭제하기 위해서는 단순히 Null로 지정

Variable

변수에 접근

정의	의미
$var	var를 출력
${var}	변수이름 일부로 번역
${var-word}	만약 설정되어 있으면 var값을, 그렇지 않으면 word값으로 대치
${var+word}	var가 설정되었을때만 word로 대치
${var=word}	만약 설정되어 있지 않으면 var에 word를 할당, var 값으로 대치
${var?word}	var가 설정되면 var로 대치, var가 설정되지 않으면 word는 표준에러로 출력되고 셸은 종료, 만일 word가 생략되면 그 대신에 표준에러 메시지가 표시됨

Variable Cont'

"read" 명령을 사용하여, 표준 입력으로부터 변수를 읽을 수 있게 함

"-n" 옵션은 현재 입력커서가 다음 줄에 표시되는 것을 막음

사용자 변수는 보통 그 변수를 만들 때 해당 셸 에서만 액세스 될 수 있다. 따라서 다른 셸에서 사용 가능하도록 하기 위해서는 "export" 명령을 사용

변수에 값을 할당하고 사용자가 이 값을 변경되지 않도록 할 수 있는데, 이때 사용하는 명령이 바로 "readolny" 명령어 임

환경(Shell) 변수

환경변수는 셸이 수행되면 자동으로 할당되는 변수
환경변수 목록

이름	값
$0~$9	명령라인의 각 단어
$@	모든 위치 매개변수의 개별적으로 인용된 목록
$?	마지막 명령의 반환 값
$!	마지막 백그라운드 명령의 프로세스 ID
$_	내장명령어 set에 의해 할당된 현재 셸 옵션
$#	매개변수 개수
$*	모든 파라미터 문자열
$$	현재 셸의 프로세스 식별번호

주석문(Comment)

일반 프로그램 언어와 같이 셸 프로그램도 주석문을 넣기 휘하여 특수한 기호 "#"를 사용

"#"기호를 라인의 맨 처음에 두면 그 라인 전체가 주석문으로 처리되며, 명령문 뒤에 "#" 기호를 넣으면 그 기호 뒤부터 주석문으로 인식

"#"이후는 항상 주석문으로 처리

프로그램을 다른 사람이 이해하기 위해서는 이 주석문이 반드시 필요한다. 만약 주석문이 없다면 그 프로그램을 이해하는데 배 이상의 시간과 노력이 필요할 것임

연산식

Bourne 셸은 자체적으로 산술계산을 지원하지 않지만 "expr" 유틸리티를 사용하여 간단한 산술연산을 할 수 있음.

"expr"에서 제공하는 산술 연산자

연산자	의미
*, / , %	곱셈, 나눗셈, 나머지 연산
+,-	덧셈, 뺄셈
=,>,>=,<,<=,!=	비교연산자
&	논리적 논리곱(AND)
|	논리적 논리합(OR)

String Operation

문자열 연산자

연산자	의미
match	일치하는 문자열 길이 반환
substr	부 문자열 추출
index	부 문자열 위치 반환
length	문자열 길이를 반환

"test" 표현식

옵션	의미	기능
()	연산의 순서를 제어, 그룹핑	
!	부정(NOT)	논리 연산자
-a	논리곱(AND)	
-o	논리합(OR)	
-z string	string의 길이가 "0" 이면 참	문자열 연산
-n string	string의 길이가 "0" 아니면 참	
string1 = string2	string1과 string2가 같으면 참	
string1 !=string2	string1과 string2가 같지 않으면 참	
n1 -eq n2	두수가 같으면 참	숫자 연산자
n1 -ne n2	두수가 같지 않으면 참	
n1 -gt n2	n1이 크면 참	
n1 -ge n2	n1이 크거나 같으면 참	
n1 -lt n2	n1이 작으면 참	
n1 -le n2	n1이 작거나 같으면 참	
-r file	file이 존재하고 읽을 수 있으면 참	파일 연산자
-w file	file이 존재하고 기록 가능하면 참	
-x file	file이 존재하고 실행 가능하면 참	
-f file	file이 존재하고 정규 파일이면 참	
-d file	file이 존재하고 디렉토리이면 참	
-s file	file이 존재하고 비어있지 않으면 참	

조건문(if ~ else, Case)

조건문은 조건이 만족할 때에만 문장을 실행

가장 간단한 구문은 바로 "if ~ then " 이다. 그러나 "if ~ then ~ else" 또는 조건문의 변형인 "case"문처럼 다소 복잡한 문장으로 구성하여 사용

"if"문은 조건이 만족하면 "then"이하의 문을 실행. 여기서 주의해야 할 점은 "if"와 "then"은 서로 분리하여 서로 다른 라인에 나타나야 함. "if" 절이 끝났음을 알리기 위하여 끝에 "fi"를 사용

"test" 명령은 "[]"로 대치할 수 있음

"case" 문은 "if"문의 확장으로 볼 수 있으며, 몇 개의 패턴에서 대응하는 것을 찾아 그 패턴 다음의 명령들을 실행하는 다중 선택 형식을 갖음

반복문(For, While, Until)

"for ~ in" 반복은 리스트 안의 각 멤버에 대하여 명령의 집합을 한번씩 실행

"While" 문은 조건식이 참일 때에만 주어진 명령을 실행한다. 즉, 먼저 조건식을 검사하여 조건식이 참이면 주어진 명령을 실행하고, 다시 조건식을 검사하여 조건식이 거짓이면 "done" 이후의 문장을 수행.

다음으로 "until" 문이 있는데 이것은 "while" 문과 비슷하나, 어떤 조건이 만족할 때까지 명령들을 반복한다. 가장 큰 차이점은 언제 중단하느냐에 있다. 즉, "while"문은 반복이 계속될 조건을 나타내는데 반하여 "until"문은 반복을 끝낼 조건들을 나타냄

분기명령(Break, Continue)

"break" 명령은 반복에서 완전히 빠져 나와 "done"절 뒤의 문장으로 실행을 옮김. 이것은 원칙적으로 반복문은 조건이 만족할 때까지만 반복을 해야하는데, 중간에 이 반복문을 빠져 나올 수 있도록 하기 위하여 사용

"Continue" 명령은 반복하는 도중에 이 명령어 다음은 수행하지 않고 반복의 처음부분으로, 즉 조건을 검색하는 조건문으로 실행을 옮김

Script Example (1)

showmethe1.sh

```
#Display # of user currently logged in
#
who | wc -l
```

Script Example (2)

showmethe2.sh

```
#Display the current date and time
## of user logged in and working Dir.
date
who | wc -l
pwd
```

Script Example (3)

showmethe3.sh

```
echo
echo “Date and Time : ”
date
echo “Number of users on the system :”
who | wc –l
echo “Your current direcotry:”
pwd
echo
date
who | wc –l
pwd
```

Script Example (4)

Iknowwhatyou.sh

```
echo
echo “Enter your name :”
read name
echo “Your name is $name !!”
echo
```

Script Example (5)

Iknowwhatyou#2.sh

```
echo
echo “Type in a long sentence : ₩n”
Read Word1 Word2 Rest
echo
echo “You See? Show your respect! ^^”
echo
```

Script Example (6)

Loop#1.sh

```
echo
echo for count in 1 2 3
do
   echo “in the for “count times”
done
echo
exit 0
```

Script Example (7)

Loop#2.sh

```
echo
carryon=Y
while [$carryon=Y]
do
   echo “I’m doing the job as long as you type Y”
   read carryon
done
echo “Job Done! Thanks”
echo
exit 0
```

Script Example (7)

Loop#3.sh

```
# Let me know if “kitri” is on the system
echo
until who | grep “$1” > ₩dev₩null
do
   sleep 30
done
echo “₩07₩07$1 is on the system now!”
echo
exit 0
```

Script Example (6)

goingaround.sh

```
$ set One Two Three
$ echo $1 $2 $3
```