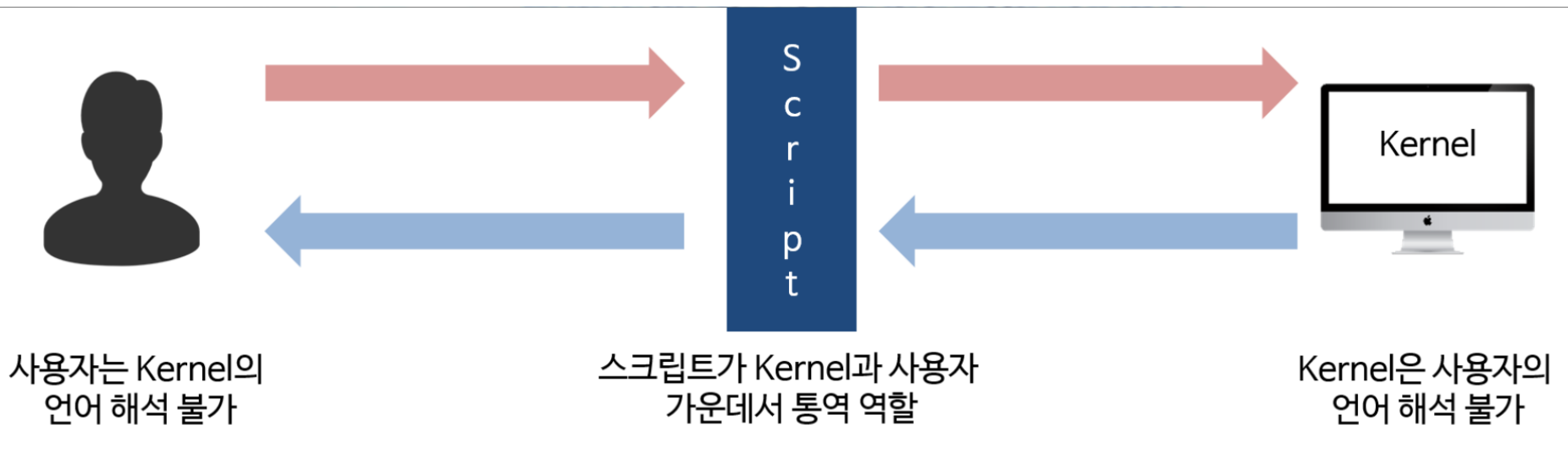


셸 프로그래밍 1

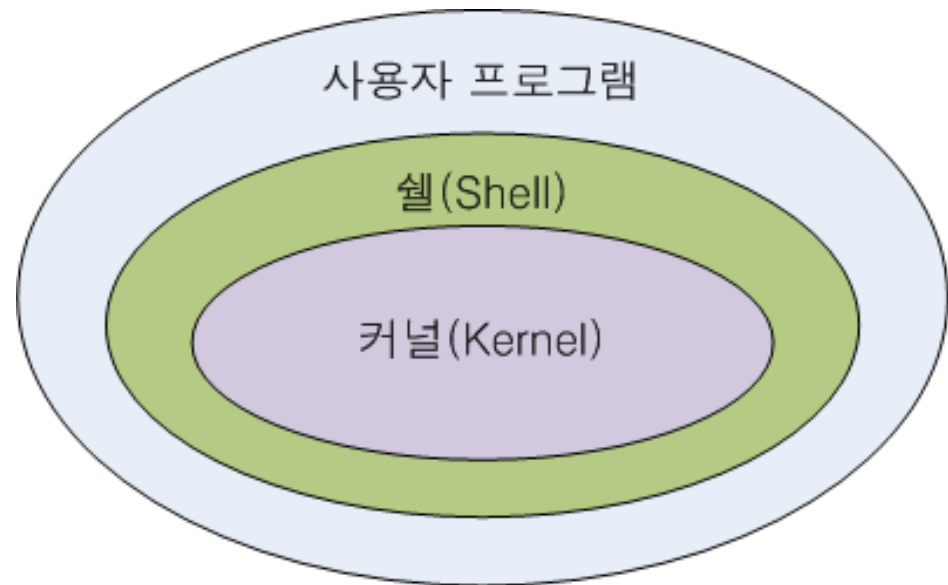
□ 셸 (Shell)

- 커널과 사용자 프로그램의 중간에 위치하는 명령어 해석기
- 키보드로부터 명령을 입력받아 운영체제(OS)가 그 명령을 수행하도록 하는 프로그램



□ 리눅스의 구성

- 리눅스 운영체제는 크게 커널, 셸, 사용자 프로그램으로 분류
 - 커널(Kernel)
 - 셸(Shell)
 - 사용자 프로그램



셸 프로그래밍 개요

□ 리눅스의 구성 –cont`d

■ Kernel

- 운영체제의 핵심
- 프로세스 관리, 메모리 관리, 파일 시스템 관리, 장치 관리
- 컴퓨터의 모든 자원을 초기화하고 제어하는 기능

■ Shell

- 커널과 사용자 프로그램의 중간에 위치하는 명령어 해석기
- 셸은 커널과 직접적으로 연결
- 사용자가 로그인을 하면 자동으로 형성되어 명령어를 해석할 수 있는 상태로 만듦 (bashrc)
- 사용자 프로그램에서 실행시킨 명령어를 해석하여 그 결과를 커널로 보내주는 역할을 함
- 셸은 종류는 다양하나, 리눅스에서는 주로 Bash(Bourne-Again Shell)을 사용

셸 프로그래밍 개요

□ 리눅스의 구성 -cont`d

■ 사용자 프로그램

- 일반적으로 리눅스에서 사용하는 프로그램
- 프로그래밍 개발도구, 문서 편집 도구, 네트워크 관련 도구 등
- 네트워크 서비스
 - 아파치 웹서버
 - FTP
 - SSH
 - 메일서버
- 각종 소프트웨어
 - firefox, openoffice 등

셸 프로그래밍 개요

□ 리눅스 셸(Linux Shell)

- 사용자의 명령어를 해석하여 수행하는 사용자 인터페이스 프로그램
- 셸의 기능
 - 명령어 해석기 기능
 - 프로그래밍 기능
 - 사용자 환경설정 기능
- 리눅스는 기본적으로 **Bash** 설정
- 다른 셸을 사용하고 싶다면 작업 중에 셸을 바꾸어 사용할 수도 있음

셸 프로그래밍 개요

□ 셸의 종류

- Bourne Shell (/bin/sh)
- Korn Shell (/bin/ksh)
- C Shell (/bin/csh)
- TC Shell (/bin/tcsh)
- Bourne Again Shell (/bin/bash)

□ 셸의 종류 – cont`d

■ Bourne Shell(sh)

- 1979년, Bell 연구소의 Steven Bourne이 개발한 최초의 대중화된 유닉스 셸
- 가장 오랫동안 UNIX 시스템의 표준 구성요소
- 후에 개발된 셸에 비해 기능적인 면에서 부족
- 현재 리눅스에서 가장 많이 사용하고 있는 Bourne Shell의 변종인 Bourne Again Shell이 있음

□ 셸의 종류 – **cont`d**

■ Korn Shell(ksh)

- 1980년대 중반 AT&T 사의 David Korn이 제작
- Bourne Shell을 포함하는 더욱 강력한 셸
- C shell과 달리 Bourne shell과의 호환성을 유지
- 히스토리 기능, 앨리어스 기능 등 C Shell의 특징을 모두 제공하면서 처리 속도도 빠르다는 장점을 가짐
- 강력한 명령어 편집기로 인해서 일반적으로 유닉스에서 많이 쓰임

셸 프로그래밍 개요

□ 셸의 종류 – cont`d

■ C Shell(csh)

- 버클리 대학교에서 개발됨
- 셸 스크립트 작성을 위한 구문 형식이 C 언어와 같아 셸의 이름도 C Shell이 되었음
- Bourne Shell과 호환이 되지만 인터페이스 구조가 다름
- C Shell은 명령어 모드에서 다양한 기능을 제공
- 유닉스 실행시의 기본 Shell로 주로 쓰이기도 함

셸 프로그래밍 개요

□ 셸의 종류 – **cont`d**

■ TC Shell(tcsh)

- 1980년대 초반, Ken Greer 등이 C Shell을 확장시켜 제작
- 프리 소프트웨어로 많은 사람들이 사용
- C Shell의 모든 기능을 제공
- 명령어 편집을 **emacs** 스타일 방식을 취함

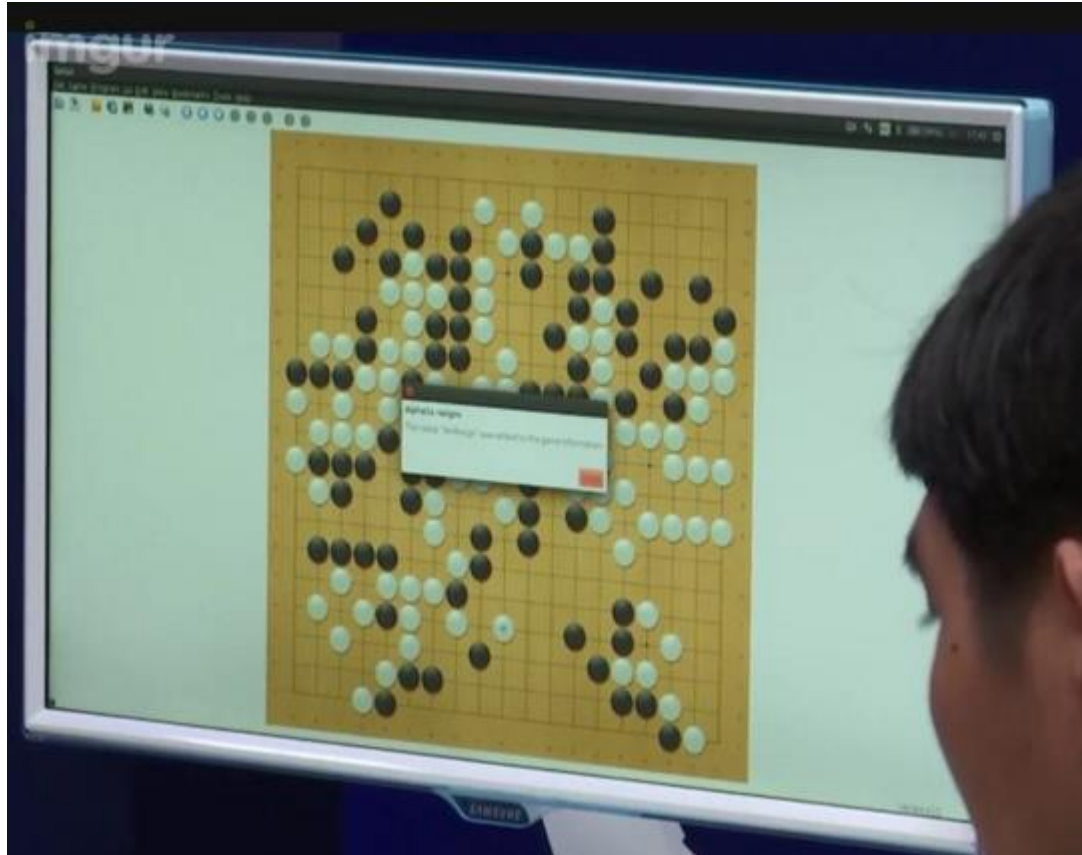
셸 프로그래밍 개요

□ 셸의 종류 – **cont`d**

■ Bourne Again Shell(bash)

- 1988년 브라이언 폭스(Brain Fox)에 의해 개발
- 리눅스의 기본 셸로 동작하고 있어 리눅스 셸로도 알려짐
- 가장 많이 이용
- Bourne Shell의 기능을 추가, 발전
- C Shell과 Korn Shell에서 제공하는 기능들도 상당수 포함
- GPL을 따르는 공개 소프트웨어

셸 프로그래밍 개요



- <http://techcrunch.com/2016/03/30/be-very-afraid-hell-has-frozen-over-bash-is-coming-to-windows-10/>

셸 프로그래밍 개요

□ 셸의 종류 – cont`d

Subject	Shell	UNIX			Linux
		Bourne Shell	Korn Shell	C Shell	Bourne Again Shell
File Name		/bin/sh	/bin/ksh	/bin/csh	/bin/bash
Prompt		\$	\$	%	\$
Reference File		.profile	.kshrc	.cshrc	.bashrc
Bourne Shell Syntax		Yes	Yes	No	Yes
Job Control		No	Yes	Yes	Yes
History List		No	Yes	Yes	Yes
Command-Line Editing		No	Yes	Yes	Yes
Aliases		No	Yes	Yes	Yes
Overwriting Protection (noclobber)		No	Yes	Yes	Yes
Setting to Ignore Control-D (ignoreeof)		No	Yes	Yes	Yes
Initialization File Separate from profile		No	Yes	Yes	Yes
Log-Out File		No	No	Yes	Yes

셸 프로그래밍 개요

□ 셸의 실행

- 로그인하면 셸 프로그램이 실행되고, 셸 프롬프트를 보여줌

□ 디폴트 셸 프롬프트

- Bourne, Korn shell : \$, #(root인 경우)
- C shell : %

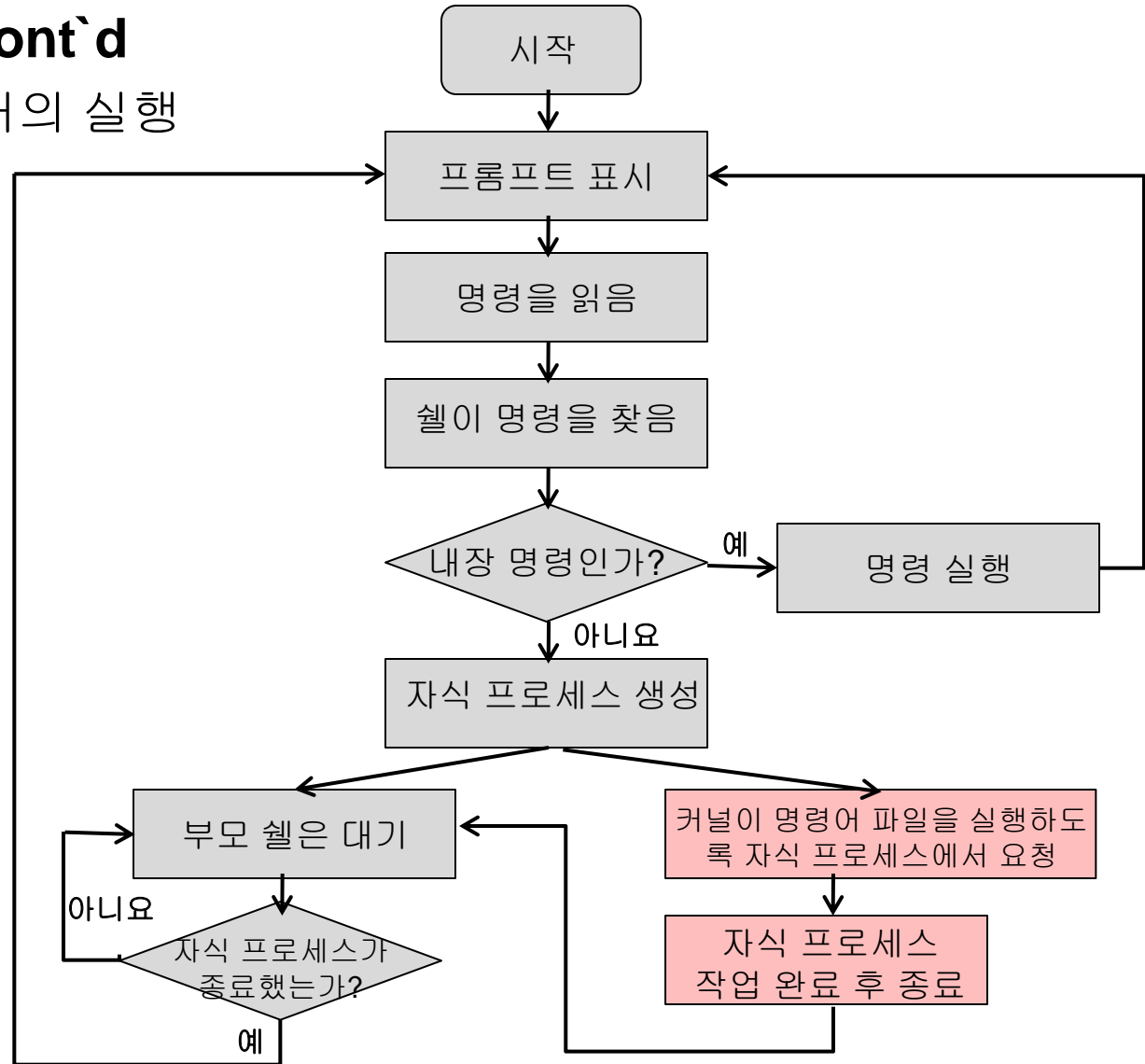
□ 사용자가 셸 프롬프트의 모양을 바꿀 수 있음

- set prompt="!%" (C shell 인 경우)
 - PS1="[u]#" (Bash shell 인 경우)
-

셸 프로그래밍 개요

□ 셸의 실행 – cont`d

■ 셸과 명령어의 실행



셸 프로그래밍 개요

- 시스템에서 사용 가능한 셸의 확인

```
[root@localhost ~]$ chsh -l or cat /etc/shells
```

- 현재 사용 중인 셸의 확인

```
[root@localhost ~]$ echo $SHELL
```

- 셸 내부 명령어 보기 (개수 확인 -nl)

```
[root@localhost ~]$ enable | nl
```

셸 프로그래밍 개요

□ 사용 셸의 변경

```
[root@localhost ~]$ chsh
```

셸 변경을 위
한 명령어

```
Changing shell for root.
```

```
New shell [/bin/bash] : /bin/sh
```

변경될 셸의
절대 경로명

```
Shell changed.
```

```
[root@localhost ~]$ logout
```

logout하고 다시
login해야 셸이
변경됨

Bash Shell

□ 환경 설정 파일

- login하여 bash가 처음 시작할 때(login shell일때), 다음 순서대로 스크립트 파일들을 수행하여 환경을 설정

- ① /etc/profile
(시스템 전반적인 환경설정 / 유저 로그인 시 시스템 초기화)
- ② ~/.bash_profile or ~/.bash_login or ~/.profile
(유저의 홈 디렉토리에 디폴트로 갖고 있는 파일/
사용자별 환경설정, 셋 중 하나가 실행됨, 주로 .profile)
- ③ ~/.bashrc
(~/.profile에서 존재여부 확인 후 호출)
(alias 설정 및 /etc/bashrc 파일 존재여부를 확인하여 실행)
- ④ /etc/bash.bashrc
(시스템 전역 함수들과 alias 설정)

- login shell이 logout 할 때 다음 파일을 찾아서 수행함

- ~/.bash_logout

- ~/ 는 로그인 사용자의 홈 디렉토리를 의미함
- login shell : login할 때 실행된 셸

Bash Shell

□ 기타 환경 설정

■ /etc/profile.d

- 터미널을 사용할 때 파일 및 디렉토리에 대한 색상 설정, 사용 언어 설정, **which** 명령어에 대한 사용자 설정값 등의 설정 파일이 있는 디렉토리

■ /etc/skel

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

□ Built-in command (내부 명령어)

- 쉘 프로그램 자체가 내부적으로 처리하는 명령어

명령어	내 용
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

□ Built-in command (내부 명령어) – cont`d

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

- 기타 명령어

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

Bash Shell

□ Built-in command (내부 명령어) – cont`d

■ echo

- 변수나 문자열을 출력
- 변수이름 앞에는 \$를 붙임
- 문자열에 공백이 포함된 경우 “ ” 사용
- echo 사용 예

```
[root@localhost ~]$ name="Kwang Woon"
[root@localhost ~]$ echo $name
Kwang Woon
[root@localhost ~]$
```

Bash Shell

□ Built-in command (내부 명령어) – cont`d

■ echo 색깔 출력

□ ANSI Escape code을 사용함

■ `^[code_numberm`

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

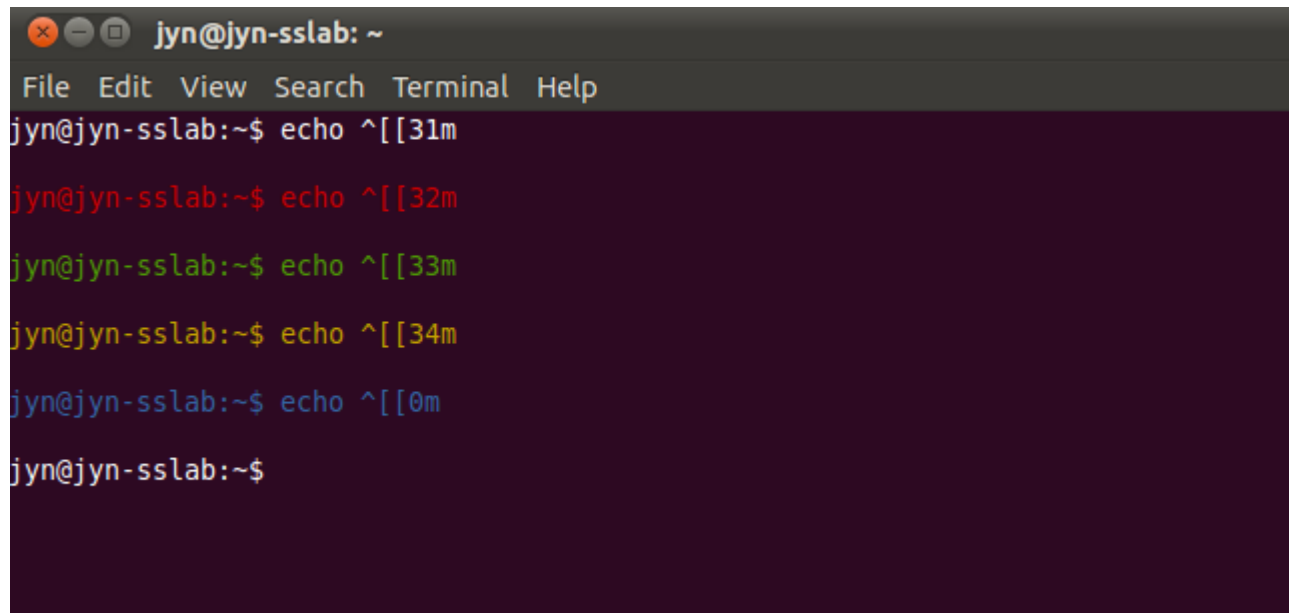
■ `^[` 입력 방법 : Ctrl-v -> v띠고 [-> 다 띠고 [

Bash Shell

❑ Built-in command (내부 명령어) – cont`d

■ echo 색깔 출력 – cont`d

❑ ^[[입력 방법 : Ctrl-v -> v 띄고 [-> 다 띄고 [



```
jyn@jyn-sslalab: ~  
File Edit View Search Terminal Help  
jyn@jyn-sslalab:~$ echo ^[[31m  
jyn@jyn-sslalab:~$ echo ^[[32m  
jyn@jyn-sslalab:~$ echo ^[[33m  
jyn@jyn-sslalab:~$ echo ^[[34m  
jyn@jyn-sslalab:~$ echo ^[[0m  
jyn@jyn-sslalab:~$
```

Bash Shell

□ Built-in command (내부 명령어) – cont`d

■ read

□ 변수에 문자열을 할당

■ 옵션

옵션	설 명
-t	사용자로부터 입력 받을 시간을 입력
-p	입력 프롬프트의 형태를 지정하며 문자열을 이용하여 출력
-a	입력 받은 변수가 배열임을 정의
-n	입력 받을 값의 크기 설정(문자의 개수)
-d	지정한 문자열을 입력 할 때까지 입력 받음
-r	입력한 이스케이프 문자를 인식
-s	사용자로부터 입력 받은 문자를 프롬프트에 echo를 하지 않기 때문에 주로 패스워드 입력에 이용

Bash Shell

□ read 사용 예

```
[root@localhost ~]$ read var_value
Linux is interesting
[root@localhost ~]$ echo $var_value
Linux is interesting
[root@localhost ~]$
```

```
[root@localhost ~]$ read -p "Input Name : " name_value
Input Name : samba
[root@localhost ~]$ echo $name_value
samba
[root@localhost ~]$
```

Bash Shell

□ Built-in command (내부 명령어) – cont`d

- **alias** : 명령어와 옵션의 조합을 다른 별칭으로 만들
 - 선언형식
 - **alias name=command**
 - **alias** 선언 예
 - **alias lf='ls -F'**
 - **alias ls='ls -l'**
 - **alias rm='rm -i'**
- **unalias** : **alias** 정의를 삭제
 - 선언형식
 - **unalias name**
- **alias** 명령어를 매개변수 없이 수행하면, 정의되어 있는 **alias** 목록이 출력됨

```
[root@localhost ~]$ alias
```

Bash Shell

□ 옵션(Option)

■ 쉘의 옵션

- 쉘의 동작을 제어하기 위해 제공되는 기능으로 **set** 명령어를 사용하여 설정할 수 있음

□ 예) optionname

- ignoreeof : ctrl+d 를 사용하여 logout 하는 것을 방지함
- noclobber : >를 사용하여 기존 파일에 덮어 쓰는 것을 방지함
- noglob : 메타문자를 순수하게 일반 문자로 인식함

■ set 명령어

- 옵션을 켤 때 : **set -o optionname**
 - 옵션을 끄 때 : **set +o optionname**
 - 옵션들의 상태를 보려면 : **set -o**
-

Bash Shell

□ 옵션(Option) – cont`d

■ set -o 예제

```
jyn@jyn-sslalab: ~  
File Edit View Search Terminal Help  
jyn@jyn-sslalab:~$ cat > aa  
sample text  
jyn@jyn-sslalab:~$ cat aa  
sample text  
jyn@jyn-sslalab:~$ set -o | grep noclobber  
noclobber      off  
jyn@jyn-sslalab:~$ set -o noclobber  
jyn@jyn-sslalab:~$ set -o | grep noclobber  
noclobber      on  
jyn@jyn-sslalab:~$ cat > aa  
bash: aa: cannot overwrite existing file  
jyn@jyn-sslalab:~$ set +o noclobber  
jyn@jyn-sslalab:~$ set -o | grep noclobber  
noclobber      off  
jyn@jyn-sslalab:~$ cat > aa  
sample text2  
jyn@jyn-sslalab:~$ cat aa  
sample text2  
jyn@jyn-sslalab:~$
```

□ 셸 변수

- 셸 변수는 시스템 변수와 사용자 정의 변수가 있음

- 변수의 값 설정

□ 변수명=변수값

주의

- =을 붙여서 사용
- = 사이에는 공백 불가

```
[root@localhost ~]$ JAVA_PATH=/tmp/java  
[root@localhost ~]$ echo $JAVA_PATH  
/tmp/java
```

- 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.

□ 셸 변수 – cont`d

■ 시스템 변수

- 시스템에서 정의한 셸 변수 (부팅할 때 설정파일에서 정의함 - /etc/profile)
- 대부분 읽기전용으로 변경불가이지만, 어떤 변수들은 사용자가 변경 가능
- **set, env, printenv** 명령어 등으로 확인 가능
- 종류

변수 이름	내 용
PS1	셸의 기본 프롬프트
PS2	셸의 하위 프롬프트
HOME	홈 디렉토리
PATH	명령어의 경로 지정
USER	사용자 이름
SHELL	셸 프로그램 파일이 위치하는 디렉토리 경로

Bash Shell

□ 셸 변수 – cont`d

■ 시스템 변수

□ 종류

변수 이름	내 용
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	bash를 시작하여 경과된 시간
IGNOREEOF	EOF가 입력되었을 때 셸의 종료를 제어
UID	사용자의 UID 값
GID	사용자의 GID 값

Bash Shell

□ 셸 변수 – cont`d

■ PATH 변수

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

□ 예)

- 기존 PATH 변수에 /home/samba/bin을 추가하는 경우

```
PATH=$PATH"/home/samba/bin"
```

Bash Shell

□ 셸 변수 – cont`d

■ PS1 변수 : 프롬프트 모양

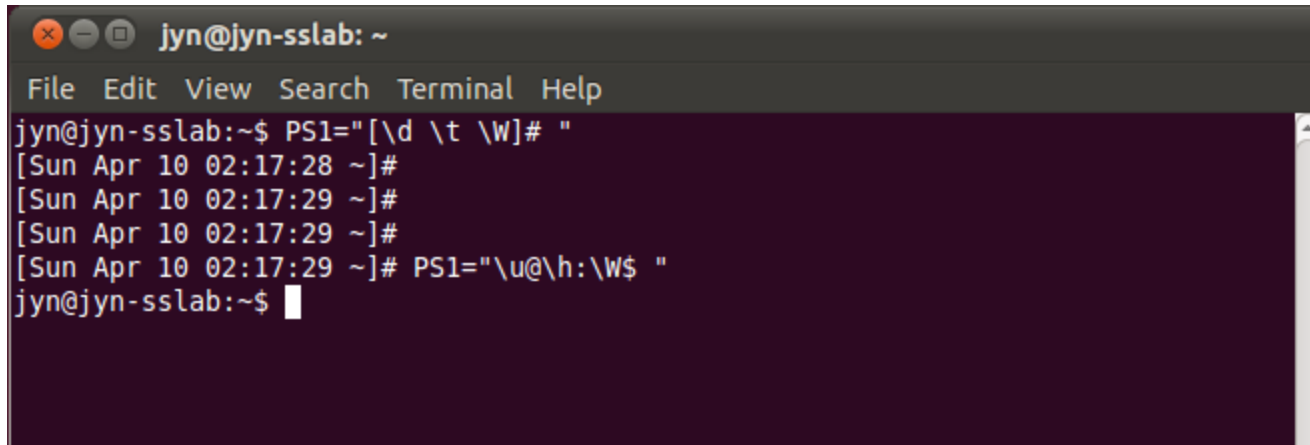
변수 이름	내 용
\d	“요일 달 날짜” 형식의 날짜 표시
\H	도메인 이름
\h	첫번째 “.”까지의 호스트 이름
\s	셸의 이름
\u	사용자 이름
\w	현재 작업 디렉토리의 절대 경로명
\W	현재 작업 디렉토리의 절대 경로명 중 마지막 디렉토리 명
\t	24-시간 형식으로 현재 시각 (HH:MM:SS)
\T	12-시간 형식으로 현재 시각
\@	12-시간으로 현재 시각, 오전/오후
\!	현재 명령어의 히스토리 번호
\#	현재 명령어의 번호
\\$	현재 UID가 0이면 #를, 아니면 \$를 표시

Bash Shell

□ 셸 변수 – `cont`d`

■ PS1 변수 : 프롬프트 모양

- `~/.bashrc`에 기록하면 로그인할 때마다 설정됨
- 프롬프트 변경 예



```
jyn@jyn-sslabs: ~  
File Edit View Search Terminal Help  
jyn@jyn-sslabs:~$ PS1="[ \d \t \W]# "  
[Sun Apr 10 02:17:28 ~]#  
[Sun Apr 10 02:17:29 ~]#  
[Sun Apr 10 02:17:29 ~]#  
[Sun Apr 10 02:17:29 ~]# PS1="\u@\h:\W$ "  
jyn@jyn-sslabs:~$
```

Bash Shell

□ 셸 변수 – cont`d

■ 사용자 정의 변수

□ 사용자가 정의하는 변수

```
[root@localhost ~]$ my_name=samba  
[root@localhost ~]$ echo $my_name  
samba
```

□ 공백이 있는 변수를 정의할 때는 “ ” 또는 ‘ ’를 사용하여 묶어줌

```
[root@localhost ~]$ my_name="genius samba"  
[root@localhost ~]$ echo $my_name  
genius samba
```

□ 셸 변수 – cont`d

■ 사용자 정의 변수

□ 명령어의 실행 결과를 변수로 지정하려면

■ Back quote(`)를 사용

```
[root@localhost ~]$ result=`cat /etc/passwd`  
[root@localhost ~]$ echo $result  
.....
```

□ 변수 값의 제거

```
[root@localhost ~]$ my_name="samba"  
[root@localhost ~]$ echo $my_name  
samba  
[root@localhost ~]$ unset my_name  
[root@localhost ~]$ echo $my_name  
  
[root@localhost ~]$
```

□ 인용부호

- 쉘에는 특별한 의미를 가지고 있는 특수문자들이 있는데, 이를 메타 문자(meta character)라고도 한다. 이러한 특수문자를 문자 그대로 사용하고자 할 때. 즉, 문자나 단어의 특별한 의미를 없앨 때 인용부호를 사용할 수 있다.

변수 이름	내 용
single quotes(' ')	문자열로 표기(변수 명 자체를 문자열로 표기)
double quotes(" ")	문자열로 표기(변수 값을 하나의 인자로 인식, 즉 \$, `, \는 문자열에서 제외됨)
backslash(\)	문자의 특수한 기능을 제거
back quotes(` `)	명령어로 인식

Bash Shell

□ 셸 변수 – cont`d

■ 메타 문자

- 셸이 특수하게 처리하는 문자
- 메타문자로 취급하지 않게 하려면 문자 앞에 \를 붙여줌
- Redirection → > < >>
- 파일 이름 대치 → * ? []
- 파이프 → |
- 명령어 대치 → Back quote(`)
- 명령어 열 → ; &&
- 명령어 그룹 → ()
- 백그라운드 실행 → &

실습 예제

```
jyn@jyn-sslalab: ~ test.sh
File Edit View Search Terminal Help
1 #!/bin/bash
2
3 echo 'My Home Directory is $HOME.'
4
5 echo "My Home Directory is $HOME."
6
7 echo "My Home Directory is \ $HOME."
8
9 echo 'My Home Directory is \ $HOME.'
10
11
12 echo "Current Directory is `pwd`."
~
~
~
```

\$HOME을 문자취급 한다

\$HOME의 값을 출력한다

\ 의 기능을 제거, \$HOME을 문자취급 한다

`는 명령어의 기능을 인정한다

```
jyn@jyn-sslalab:~$ vi test.sh
jyn@jyn-sslalab:~$ chmod 755 test.sh
jyn@jyn-sslalab:~$ sh test.sh
My Home Directory is $HOME.
My Home Directory is /home/jyn.
My Home Directory is $HOME.
My Home Directory is \ $HOME.
Current Directory is /home/jyn.
jyn@jyn-sslalab:~$
```

Bash Shell

□ 셸 변수 – cont`d

- export 명령어 : 변수를 환경변수(시스템 변수)로 만듦

방법①

```
[root@localhost ~]$ COLOR=red  
[root@localhost ~]$ export COLOR
```

방법②

```
[root@localhost ~]$ export COLOR=red  
[root@localhost ~]$ export -n COLOR
```

COLOR를
환경변수에
추가

그냥 export를 하면 환경변수를 볼 수 있음
\$ export

set -a나 set -o allexport를 하면
이후에 정의하는 모든 변수를 환경변수로 만듦

COLOR를 -n
플래그를 이용
환경변수에서
제거

Bash Shell

□ 쉘 변수 – cont`d

■ export 명령어

```
jyn@jyn-sslabs: ~  
File Edit View Search Terminal Help  
jyn@jyn-sslabs:~$ export | grep US  
declare -x DBUS_SESSION_BUS_ADDRESS="unix:abstract=/tmp/dbus-28qC3U0oCI,guid=911  
7518b435bef5c8e315eba00000015"  
declare -x GDM_LANG="en_US.UTF-8"  
declare -x LANG="en_US.UTF-8"  
declare -x USER="jyn"  
declare -x USERNAME="jyn"  
jyn@jyn-sslabs:~$ export USER_PATH="/home/game"  
jyn@jyn-sslabs:~$ export | grep US  
declare -x DBUS_SESSION_BUS_ADDRESS="unix:abstract=/tmp/dbus-28qC3U0oCI,guid=911  
7518b435bef5c8e315eba00000015"  
declare -x GDM_LANG="en_US.UTF-8"  
declare -x LANG="en_US.UTF-8"  
declare -x USER="jyn"  
declare -x USERNAME="jyn"  
declare -x USER_PATH="/home/game"  
jyn@jyn-sslabs:~$ export -n USER_PATH  
jyn@jyn-sslabs:~$ export | grep US  
declare -x DBUS_SESSION_BUS_ADDRESS="unix:abstract=/tmp/dbus-28qC3U0oCI,guid=911  
7518b435bef5c8e315eba00000015"  
declare -x GDM_LANG="en_US.UTF-8"  
declare -x LANG="en_US.UTF-8"  
declare -x USER="jyn"  
declare -x USERNAME="jyn"  
jyn@jyn-sslabs:~$
```

Bash Shell

□ 셸 변수 – cont`d

■ 환경(environment)

- 셸에 의해 초기화되어 export된 셸 변수들
- 셸의 자식 프로세스(셸에 의해 수행되는 프로그램)에게 상속되어 전해짐

■ 환경에 변수를 추가하기 위한 2 가지 방법 (즉, 환경변수로 만들려면)

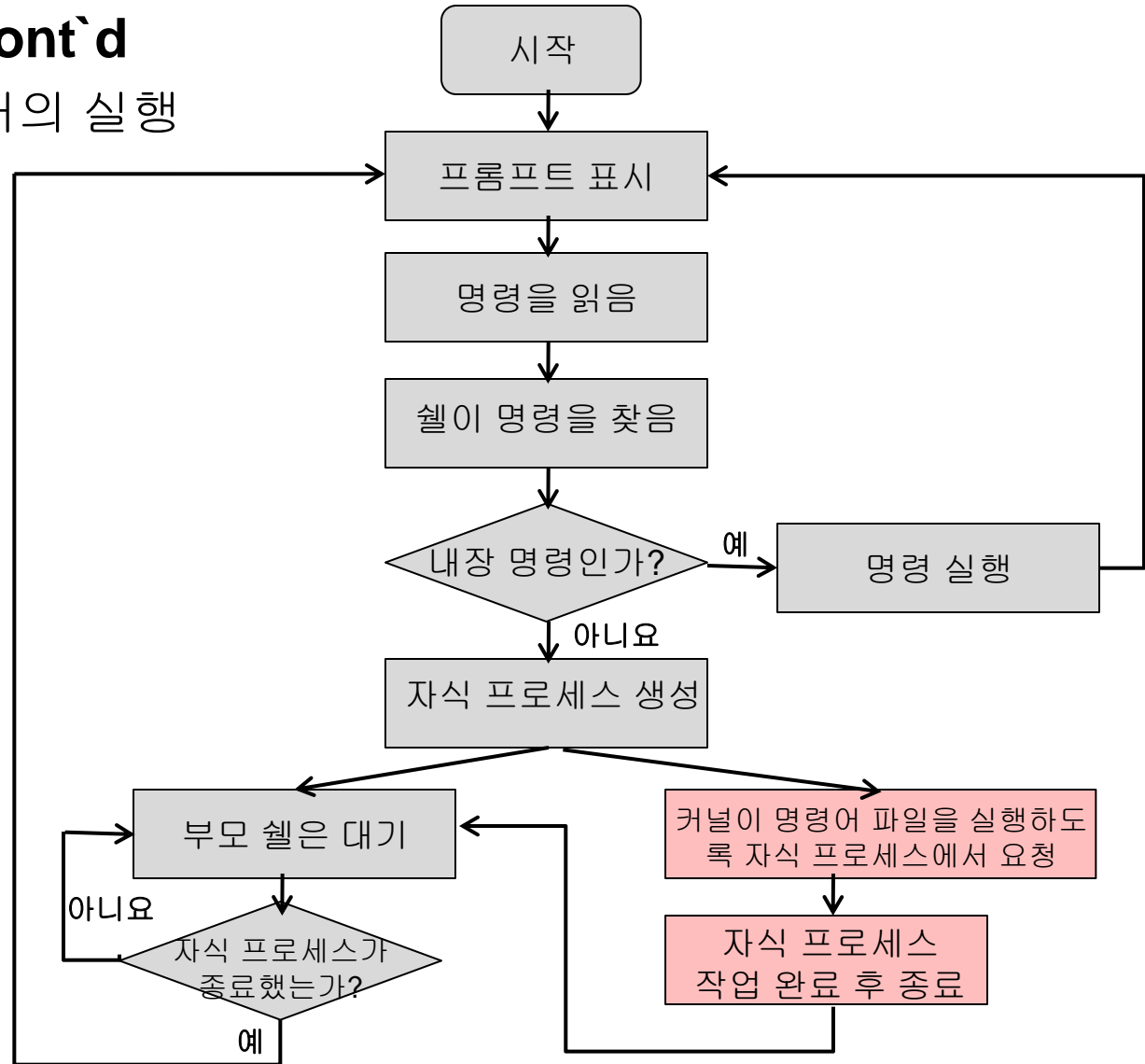
- export 변수이름=값
- declare -x 변수이름=값

■ 환경 변수가 아닌 것은 셸의 자식 프로세스에 상속되지 않음

셸 프로그래밍 개요

□ 셸의 실행 – cont`d

■ 셸과 명령어의 실행



Bash Shell

□ 셸 변수 – cont`d

■ 환경의 상속 예

상속안됨

상속됨

```
jyn@jyn-sslalab: ~  
File Edit View Search Terminal Help  
jyn@jyn-sslalab:~$ NAME=SAMBA  
jyn@jyn-sslalab:~$ export NAME2=CVS  
jyn@jyn-sslalab:~$ /bin/bash  
jyn@jyn-sslalab:~$ echo $NAME  
  
jyn@jyn-sslalab:~$ echo $NAME2  
CVS  
jyn@jyn-sslalab:~$ exit  
exit  
jyn@jyn-sslalab:~$ echo $NAME  
SAMBA  
jyn@jyn-sslalab:~$
```

자식 프로세스를 만듦

자식 프로세스인
/bin/bash가 수행 중

자식 프로세스 종료
부모에게 돌아감

Bash Shell

□ 셸 변수 – cont`d

- 특별 내장 변수 : 셸이 내부적으로 관리하는 변수

변수 이름	내 용
\$\$	실행중인 프로세스의 PID
\$?	마지막에 실행한 명령어의 종료 값
\$!	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 인수 값