시스템 프로그래밍

주제	.bashrc 파일 분석
이름	김경환
학번	2011136008
제출일	2016 . 10 . 06

- 목차

1. 개요

1) 목적

쉘과 쉘 프로그래밍이 무엇인지 알고, .bashrc 파일을 통하여 쉘의 역할과 프로그래밍에 대하여 고찰한다.

2) 쉘과 쉘 프로그래밍에 대하여

쉘이란, 사용자와 리눅스 사이에 인터페이스를 제공하는 프로그램이다.

쉘 스크립트는 쉘이나 명령줄 인터프리터에서 돌아가도록 작성되었거나 한 운영체제를 위해 쓰인 스크립트이다. 단순한 도메인 고유 언어로 여기기도 한다. 쉘 프로그래밍이란, 쉘 스크립트를 쉘 언어로 작성하는 것이다.

장점으로는 , 쉘 스크립트를 기록하는 것은 다른 프로그래밍 언어의 같은 코드로 쓰인 것 보다 훨씬 빠른 경우가 많다. 다른 해석 언어에 비해, 쉘 스크립트는 컴파일 단계가 없기 때문에 스크립트는 디버깅을 하는 동안 빠르게 실행할 수 있다.

쉘 프로그래밍의 특징으로는 다음이 있다.

1) 타입이 존재하지 않는다. (모두 문자열로 취급한다.)
2) 대소문자를 구별한다.
3) 미리 선언할 필요가 없다.
4) 변수 접근은 $ 를 통해 접근한다.

2.본문

1) .bashrc

.bashrc는 대략적으로 명령어 히스토리의 유지, 쉘 프롬프트의 색상 설정, 명령어 변수를 다른 문자열로 치환하는 역할을 한다. 이제 각 요소를 살펴 보겠다.

```
case $- in
    *i*) ;;
      *) return;;
esac
```

case 변수는 다른 언어와 마찬가지로 변수를 기준으로 실행할 내용을 결정한다. Case 의 끝은 esac를 통하여 명시한다. $-는 옵션변수인데, i(interactive) 옵션이 켜져 있는지를 검사하고 켜져있지 않으면 프로그램을 끝낸다.

```
# HISTCONTROL 변수를 통해 어떻게 명령어 기록을 저장할지를 설정할 수 있다.
# ignorespace, ignoredups 변수가 있으며 각각의 의미는 공백으로 시작하는 명령어를
# 저장할지, 중복되는 명령어를 저장할지이며 둘다 설정은 아래 변수와 같다.
HISTCONTROL=ignoreboth

# 명령어 저장 기록을 덮어쓰지 않고 확장한다.
shopt -s histappend

# 명령어 히스토리 사이즈를 지정한다.
HISTSIZE=1000
HISTFILESIZE=2000

# 각 명령 후에 윈도우 사이즈를 체크하고,
# 필요하다면 열과 행의 값을 업데이트 한다.
shopt -s checkwinsize
```

쉘은 과거 입력했던 명령어를 저장하고 빠르게 다시 이용할 수 있도록 하는데, 이것에 대한 기타 정보를 유지 하는 것도 .bashrc의 역할이다.

HISTCONTROL은 3개의 상수 변수 대입이 가능하다.

1) ignorespace : 앞에 공백을 포함한 명령어를 히스토리에 저장하지 않는다.

2) Ignoredups: 중복된 명령어를 저장하지 않는다.

3) Ignoreboth : 위의 두개를 모두 적용한다.

```
# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# 당신이 작업하고 있는 곳에서 를  chroot를 식별하는 변수를 설정한다
# debian_chroot가 가르키는 값이 없고 파일이 존재하면 그 파일로 변수를 설정
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# TERM 변수를 조작하여 쉘 프롬프트 환경에서 색을 설정한다.
case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac
```

x 옵션은 파일이 실행가능한지에 대한 여부를 반환하는 것이며 &&는 앞의 test가 참을 반환해야 뒤의 명령을 실행할 수 있다는 옵션이다.

z 옵션은 debian_chroot 변수가 존재하지 않고 /etc/debian_chroot 파일이 존재하면 읽어들여서 현재 작업 디렉토리를 루트 디렉토리 처럼 인식하게 된다.

TERM 변수는 터미널 변수로서 위의 구문은 변수의 값이 xterm-color 이거나 256 color로 끝나는 것이라면 색상 프롬프트를 설정한다.

```
# (변수) 가 공백이 아닌 경우, (경로) 가 실행 가능하면서 (명령어)가 실행 가능 한
# 경우를 판단하고 color_prompt 를 설정
if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
        # We have color support; assume it's compliant with Ecma-48
        # (ISO/IEC-6429). (Lack of such support is extremely rare, and such
        # a case would tend to support setf rather than setaf.)
        color_prompt=yes
    else
        color_prompt=
    fi
fi
# color_prompt 변수에 따라, 커맨드 프롬프트 변수를 변경한다.
```

Force_color_prompt 환경변수가 null이 아니고 , 상위 경로가 실행가능한 파일이며, 명령어가 실행 가능한 경우에는 color_prompt 값을 yes로 변경한다.

>&/dev/null 은 표준 출력을 모두 폐기하는 것을 의미한다.

```
# 여기 아래 부터는 각종 긴 명령어나 자주 사용하는 명령어를
# 짧은 명령어로 치환하는 정의가 alias가 기술된다.

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

# 따로 정의된 명령어 변수를 지원한다.
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

Alias 명령어는 복잡한 명령어를 치환하는 것으로 길거나 헷갈리는 옵션이 있을 때, 이를 다른 명령어로 치환하는 기능을 한다.

.bash_aliases를 사용하면 별도의 .bashrc를 수정하는 일 없이, 사용자가 쓰기 편하게 명령어를 수정해 놓고 이용할 수 있다.

```
if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
# 만약 이 쉘에서 유저 이름과 경로를 각 라인에 표시하기로 설정 했다면 프롬프트 변수에 반영한다.
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac
```

PS1 은 커맨드 프롬프트 변수로서, 명령창 각 라인마다 사용자의 이름을 출력하는 역할을 한다.

자세한 설정 방식은 .bash_profile에서 보이겠다.

```
# 각종 변수에 alias 키워드를 통해 색상 설정을 지원한다.
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi


# 또한 쉘 명령창에서의 에러와 경고에 대하여 색상을 지원할 수도 있다.
# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'
```

Alias 명령어치환을 통해서 자주 사용하는 명령어에 색상을 부여할 수 있다.

2) bashrc_profile

PS1 변수에 특정 값을 대입함으로서, 각 커맨드라인 한줄마다 각종 정보를 출력해 줄 수 있다.

PS1을 설정하는 방법으로는

1) 현재 사용중인 쉘에 바로 사용하기

   Export PS1="[₩u@₩h₩w]₩₩$"

2) /etc/bashrc에 등록

3) ~/.bashrc에 등록

4) ~/.bashrc_profile에 등록

지속적인 변경을 위해서는 파일에다가 등록해야할 필요가 있다. 그렇다면 이제, PS1에 사용하는 변수들에 대하여 알아보자.

PS1 변수에 사용되는 기호들과 그 의미

쉘변수 기호	의미
₩t	24시간의 단위로 현재시각을 HH:MM:SS 로 표시
₩T	12시간의 단위로 현재시각을 HH:MM:SS 로 표시
₩@	12시간의 단위로 현재시각을 오전/오후 로 표시
₩d	현재 날짜를 나타냄. 요일, 월, 일 형식으로
₩s	현재 사용중인 쉘의 이름을 나타냄 (C쉘이면 /bin/csh, bash쉘이면 /bin/bash)
₩w	현재 디렉토리의 전체 절대경로를 모두 표시함
₩W	현재 디렉토리의 전체 절대경로명 중 마지막 디렉토리명만을 표시함. 즉 현재디렉토리만 표시함
₩u	사용자명을 표시함
₩h	서버의 호스트명을 표시함 (www.uzuro.com에서 www 부분)
₩H	서버의 도메인명을 표시함 (www.uzuro.com에서 uzuro.com 부분)
₩#	접속한 순간부터 사용한 명령어의 번호를 1번부터 차례대로 표시함
₩!	사용한 명령어의 history 번호를 표시함
₩₩$	현재 사용자가 root(uid 가 0)이면 # 을 표시하고 아니면 $ 를 표시함
₩₩	'₩' 문자 자체를 표시함
₩a	ASCII 종소리 문자 (07)
₩e	ASCII 의 escape 문자 (033)
₩n	개행문자 (줄바꿈)
₩v	사용중인 bash 의 버전
₩V	사용중인 bash 의 배포, 버전+패치수준으로 버전을 상세히 표시함
₩r	Carrage retrun

특히 ₩[그리고 ₩] 은 비출력 문자임에 유의한다.

```
export PS1="\e[0;31m[\u@\h \W]\$ \e[m "
```

위의 구문을 분석해 보자면, ₩e[는 색상 적용의 시작, 0;31의 색상, 즉 빨간색 적용,

표시 내용은 u h W 유저 , 사용 호스트, 워킹 디렉토리를 의미한다.

3.고찰

이번 과제를 통해 다음 내용에 대하여 알게 되었다.

- 쉘과 쉘 프로그래밍이 리눅스에서 하는 역할
- 쉘을 구성하는 if,for,while,case 등의 사용법
- [] 과 test를 구성하는 비교 옵션들
- Bashrc의 역할과 구성 명령어들
- Bash_profile을 통한 커맨드라인 출력속성 변경