배쉬(Bash) 프롬프트 HOWTO

Giles Orr, giles@interlog.com v0.55, 1 December 1998

옮긴이: 김 홍수, hskim@supy.kaist.ac.kr 옮긴 날:1998년 12월 24일

터미널과 X텀 프롬프트들을 어떻게 생성하고 다룰 것인가에 대하여 논의하며, 또한 사용자명, 현재 작업 디렉토리, 시간 등을 나타내기 위한 표준적인 escape sequence 들을 병합하는 방법도 포함합니다. X텀 제목 표시줄(title bar)을 변경하는 방법, 프롬프트에 정보를 제공하기 위한 외부 함수들의 사용 방법, ANSI 색(color)을 어떻게 사용할 것인가에 대하여 더 많은 제안 들이 제시됩니다.

제 1 절 내용 소개

1.1 요구 사항

우선, 배쉬가 필요할 것입니다. 대부분의 리눅스 배포본에서 디폴트 버전은 1.14.7 이며(98년 11월, 이 글이 씌어질 때), 이 버전은 잘 알려져 있으며 또한 신뢰할 만한 쉘입니다. 배쉬는 현재로 버전 2.0+ 까지 쓰이고 있읍니다:저는 얼마 전 부터 배쉬2.0을 사용해 오고 있읍니다만, 여기에서 제시되는 대부분의 명령어들은 버전 1.14.7 이하에서도 작동할 것입니다. 만약 문제를 인식하게 되면, 그것에 대해서 언급하도록 하겠읍니다. 프롬프트에서 'echo $BASH_VERSION' 이라고 치면 배쉬 버전을 확인할 수 있읍니다. 제 기계의 경우, '2.02.1(1)-release'라고 반응합니다.

쉘 프로그램 경험이 있으면 좋으나, 꼭 필요한 것은 아닙니다: 더 많이 알면, 더 복잡한 프롬프트들을 생성할 수 있을 것입니다. 쉘 프로그램이나 유닉스 도구들에 대한 기초적인 지식은 가지고 있다고 가정합니다. 그러나, 제 자신의 쉘 프로그램 기술이 한계가 있으므로, 고도의 쉘 프로그래머에게는 필요없어 보일 수도 있는 예제나 설명들을 많이 하게 될 것입니다.

1.2 제안 사항이나 비평 보내기

이 강의는 저에게는 배우는 경험입니다. 어떻게 하면 재미있고 유용한 프롬프트를 만들 수 있는지 많이 알게 되었지만, 이 문서의 내용을 수정하고 향상 시키는 데는 여러분들의 참여가 필요합니다. 이 문서의 가장 최근의 버전은 <http://www.interlog.com/~giles/bashprompt.html> 에서 이용할 수 있읍니다. 이 홈 페이지를 확인하시고, 자유롭게 giles@interlog.com 의 주소로 e-mail 을 주도록 하십시오.

1.3 칭찬의 글

이 문서를 만드는데는, <http://bash.current.nu/>의 배쉬프롬프트 프로젝트의 작업 으로 부터 많은 것을 차용하였읍니다. 사용된 다른 출처들은 아래와 같읍니다. Eric Lister의 'the xterm Title mini-HOWTO' <http://sunsite.unc.edu/LDP/HOWTO/mini/Xterm-Title.html>, Keebler의 'Ansi Prompts' <http://www.ncal.verio.com/~keebler/ansi.html>, Stephen Webb의 'How to make a Bash Prompt Theme' <http://bash.current.nu/bash/HOWTO.html>, Stumpy의 'X ANSI Fonts' <http://home.earthlink.net/~us5zahns/enl/ansifont.html>.

또한, 유닉스에 대한 지식에서는 저를 초월하는 Dan, Georgia College & State University에서 나의 동료 연구자, 과의 몇 번의 토론과 e-mail에서 막대한 도움을 받았읍니다. 그는 몇 번의 아주 우수한 제안과 아이디어를 제시했으며, 결과로 몇 몇의 재미있는 프롬프트를 만들게 되었읍니다.

프롬프트들을 프로그램하는데 유용하게 사용된 세권의 책이 있읍니다. Jessica Heckman Perry의 'Linux in a Nutshell'(O'Reilly, 1997), Cameron Newham 과 Bill Rosenblatt의 'Learning the Bash Shell'(O'Reilly, 2nd.

ed., 1998), Lowell Jay Arthur의 'Unix Shell Programming' (Wiley, 1986. 이 책은 첫 판이고, 네번째 판이 1997년도에 출판되었읍니다.).

제 2 절 배쉬와 배쉬 프롬프트

2.1 배쉬란 무엇인가

Bourne Shell로 부터 전승된, 배쉬는 GNU 프로젝트인 "Bourne Again SHell"이며, 대부분의 리눅스 기계에서의 표준적인 명령어 라인 인터페이스입니다. 이것은 상호연결성에서, 그리고 명령어 라인 편집, 완료, 재실행을 지원하는데 탁월합니다. 배쉬는 또한, 형성가능 프롬프트를 지원합니다-많은 사람들이 이점을 인식하고는 있지만 아주 많은 것이 가능하다는 사실을 모르고 있읍니다.

2.2 배쉬를 잡아 비틀어서 무엇을 하게 할 것인가

대부분의 리눅스 시스템은 단색(일반적으로 회색)의 디폴트 프롬프트를 가지고 있어, 사용자 명, 작업하고있는 기계의 이름, 그리고 현재의 작업 디랙토리등을 표시합니다. 이것이 유용한 정보의 전부이지만, 프롬프트로 훨씬 많은 것을 할 수 있읍니다: 모든 종류의 정보(tty번호, 시간, 날짜, 부하, 사용자의 수, 걸린 시간...)를 표시할 수 있으며, 프롬프트는 ANSI 색을 사용할 수 있어서 재미있게 보이거나, 어떤 정보 는 돋보이게 할 수 있읍니다. 또한 X텀의 제목 표시줄을 조정할 수 있어서 이들 정보 중 몇 개를 반영할 수 있도록 합니다.

2.3 고민은 그만하시오

시스템 정보의 자취를 유지하는 것은 종종 유용한 경우가 많읍니다. 몇몇의 사람들 에게 호소력 있게 보이는, 제가 아는 아이디어 중 하나는, 각각의 서로 다른 기계 마다 다른 색깔의 프롬프트를 유지하도록 할 수 있다는 것 입니다. 만약 몇 개의 다른 기계에서 X텀들을 열고 있다거나, 또는 작업 중인 기계가 어느 기계인지 잊어버려서 엉뚱한 파일들을 지워 버리는 경향이 있다면, 이 방법은 현재 작업 중인 기계를 상기 시키는데 있어서 매우 현명한 방법이라는 것을 알게 될 것입니다.

2.4 첫번째

프롬프트의 생김새는 PS1 쉘 변수로 제어합니다. 명령의 계속은 PS2 쉘 변수에 의해 지정되는데, 이것은 여기서 논의되는 바와 꼭 같은 방법으로 수정할 수 있읍니다- -PS2를 조정하는 것은 정확하게 똑같은 방법이라서, 그렇게 흥미롭지 않으므로 PS1 쉘 변수를 대부분 수정할 것입니다(일반 사용자들은 본적이 없는 PS3, PS4 쉘 변수도 있읍니다-이것들의 사용에 흥미가 있으신 분들은 배쉬 메뉴얼 페이지를 참고 하세요). 프롬프트를 다르게 보이도록 하기 위해서, PS1 쉘 변수를 바꾸면 됩니다. 실험적으로, 현재의 프롬프트에서 직접 PS1 변수를 입력해서 당장 그 결과를 보실 수 있읍니다.(이것은 현재의 쎄션에서만 효력이 있고 로그 아웃하게 되면 사라지게 됩니다). 바뀐 프롬프트를 영구적으로 유지할려고 한다면, ' /.bashrc' 파일을 수정 하여, 그곳에 PS1의 새로운 정의를 첨가하도록 하십시오. 만약 루트의 권한을 갖고 계시다면 '/etc/profile'이라는 파일을 찾아 "PS1=" 라인을 수정할 수 있읍니다. 몇몇의 배포본(적어도 RedHat 5.1)에서는 '/etc/bashrc'파일이 PS1, PS2 쉘 변수를 무효화 시킨다는 점을 주의 하십시오.

시작하기 전에, PS1 변수는 다른 환경 변수들과 마찬가지로 환경변수 기억 장소에 저장되어 있다는 것을 명심해야 합니다. 명령어 라인에서 PS1을 수정하게되면, 당장 프롬프트가 바뀌게 됩니다. 어떠한 변경을 가하기 전에, 또다른 환경변수에 현재의 프롬프트를 저장할 수 있읍니다.

```
[giles@nikola giles]$ SAVE=$PS1
[giles@nikola giles]$
```

가장 간단한 프롬프트는 다음과 같이, 하나의 문자로 될 것이죠.

```
[giles@nikola giles]$ PS1=$
$ ls
bin   mail
$
```

명령어 라인에서 입력함으로써, 기본적인 프롬프트에 대한 실험의 최상의 방법을 설명합니다. 사용자가 입력한 문장은 프롬프트의 바로 뒤에 나타난다는 점을 주목 하십시오: 저는 아래와 같이 사용하기를 더 좋아 합니다.

```
$PS1="$ "
$ ls
bin   mail
$
```

이것은 프롬프트 뒤에 공백을 두어서, 더 읽기 쉽도록 합니다. 원래의 프롬프트로 복귀하기 위해서는, 단지 저장하였던 변수를 부르기만 하면 됩니다.

```
$ PS1=$SAVE
[giles@nikola giles]$
```

2.5 배쉬 이스케잎 시퀀스(escape sequence)

배쉬 쉘에서는 프롬프트에 삽입하기 위해서 많은 이스케잎 시퀀스가 제공됩니다. 배쉬2.02 메뉴얼 페이지로 부터:

상호적으로 실행할 때, 배쉬는 명령을 받아들일 준비가 되면 기본 프롬프트 PS1을 표시하고, 명령어를 완성하기 위해 더 많은 입력이 요구될 때는 두번째 프롬프트인 PS2를 나타냅니다. 배쉬에서는 이들 프롬프트 변수들이 backslash(\)와 하나의 문자로 조합된 특수 문자들로 만들어지도록 하며, 그것들은 아래와 같이 해석 됩니다.

```
\a     ASCII 종소리 문자(07)
\d     "요일  달  날짜" 형식의 날짜 표시
      (예 : "Tue May 26")
\e     ASCII의 escape 문자 (033)
\h     첫번째 "." 까지의 hostname
\H     hostname
\n     새로운 줄(줄 바꾸기)
\r     carrage return
\s     쉘의 이름, $0 의 basename
      (마지막 slash의 다음에 따라오는 부분)
\t     24-시간으로 현재 시각, HH:MM:SS(시:분:초) 형식
\T     12-시간으로 현재 시각, HH:MM:SS(시:분:초) 형식
\@     12-시간으로 현재 시각, 오전/오후 형식
\u     현재 사용자의 username
\v     bash의 버전(예: 2.00)
\V     bash의 배포, 버전 + 패치수준
      (예 : 2.00.0)
\w     현재 작업 디랙토리
\W     현재 작업 디랙토리의 basename
\!     현재 명령어의 history 번호
\#     현재 명령어의 command 번호
```

```
\$      유효한 UID가 0 이면 # 를, 그렇지 않으면 $ 을 표시
\nnn    8 진수 nnn에 해당하는 문자
\\      backslash
\[      터미널 조정 sequence를 프롬프트에 끼워 넣기 위해
        사용될 수 있는 non-printing 문자의 시작
\]      non-printing 문자의 끝
```

떠났던 곳에서 계속해서 :

```
[giles@nikola giles]$ PS1="\u@\h \W> "
giles@nikola giles> ls
bin   mail
giles@nikola giles>
```

이것은 대부분의 배포본에서의 디폴트와 유사합니다. 저는 약간 다르게 보이기를 원해서 이렇게 바꾸었습니다:

```
giles@nikola giles> PS1="[\t][\u@\h:\w]\$ "
[21:52:01][giles@nikola:~]$ ls
bin   mail
[21:52:15][giles@nikola:~]$
```

제 3 절 외부 명령어들

3.1 PROMPT_COMMAND

배쉬에서는 또 다른 환경 변수 PROMPT_COMMAND가 제공됩니다. 배쉬가 프롬프트를 표시하기 바로 직전에, 정상적인 배쉬 명령어로서, 이 변수의 내용이 실행됩니다.

```
[21:55:01][giles@nikola:~] PS1="[\u@\h:\w]\$ "
[giles@nikola:~] PROMPT_COMMAND="date +%H%M"
2155
[giles@nikola:~] d
bin   mail
2156
[giles@nikola:~]
```

위에서 일어난 일을 말씀드리자면, 이스케잎 시퀀스에서 '\t'를 제외 시켜서, 프 롬프트에 시각이 더 이상 포함되지 않도록 하였읍니다. 그리고나서 'date +%H%M' 를 사용하여 제가 더 좋아하는 형식으로 시각을 표시하도록 하였읍니다. 그렇지만 이번에는 프롬프트와는 서로 다른 줄에 나타났읍니다. 배쉬2.0+ 에서는 'echo -n ...'를 사용하여 잘 정돈할 수 있으며, 아래에서와 같이 작용하지만, 배쉬1.14.7에서는 작동하지 않게 됩니다.: 명백히 프롬프트는 다른 식으로 작성되 고, 아래와 같은 방법은 중복된 문장으로 나타납니다.

```
2156
[giles@nikola:~] PROMPT_COMMAND="echo -n [$(date +%H%M)]"
[2156][giles@nikola:~]$
[2156][giles@nikola:~]$ d
bin   mail
[2157][giles@nikola:~]$ unset PROMPT_COMMAND
[giles@nikola:~]
```

'echo -n ...'에 의해 날짜 명령의 출력이 조절되고 뒤에 따르는 '새줄' 문자가 무효 화되어, 프롬프트와 함께 모두 한 줄에 나타나게 됩니다. 마지막에서, 'unset' 명령 에 의해 환경 변수 PROMPT_COMMAND가 제거되었읍니다. 명령어 대치를 위해 $() 양식이 사용된 것에 유의하십시오: 즉,

```
$(date +%H%M)
```

는 "여기에다 'date +%H%M' 명령으로 부터의 출력으로 대치하시오" 라는 의미 입니다. 이것은 배쉬2.0+ 에서 작동합니다. 1.14.7 이전의 몇몇 구 버전의 배쉬 에서는 역인용 부호(backquates)('date +%H%M')가 필요할 수가 있읍니다. 역인용 부호는 배쉬2.0+ 에서도 사용 가능하지만, $() 양식을 더 선호 함에 따라 사라져 가고 있으며, $() 양식이 자리를 굳혀 가고 있읍니다. 이 문서에서도 새 양식으로 계속 될 것 입니다. 만약 이전 버전을 사용하고 계시다면, $() 이 나타나는 곳에서 역인용 부호로 대치하면 될 것입니다. 만약, 명령어 대치가 이스케잎 처리되어 있다면 (즉, \$(command)), basckslash('\')를 사용하여 양쪽의 역인용 부호를 이스케잎 처리 하십시오(즉, \'command\').

3.2 프롬프트에서의 외부 명령어들

프롬프트에서 정상적인 리눅스 명령의 출력 결과도 역시 사용 가능합니다. 너무 많은 것들을 프롬프트에 포함하기를 원하지는 않을 것입니다. 그렇지 않으면 프롬프트가 너무 길어질 테니까요. 또한, 신속히 처리되는 명령어를 사용하기를 원할 것입니다. 왜냐하면, 스크린에 프롬프트가 나타날 때 마다 그명령어들이 수행되어야 하고, 작업을 하는 동안에 프롬프트가 나타나는 것이 성가시게 될 수도 있기 때문입니다.(이전의 예와 아주 유사한 결과이지만, 배쉬1.14.7에서도 작동합니다)

```
[21:58:33][giles@nikola:~]$ PS1="[\$(date +%H%M)][\u@\h:\w]\$ "
[2159][giles@nikola:~]$ ls
bin   mail
[2200][giles@nikola:~]$
```

명령어 대치의 달러기호('$') 앞의 backslash에 주의하는 것이 중요합니다. 그것이 없으면 외부 명령어는 정확하게 한번만 수행됩니다: PS1 변수가 읽어 들여질 때 마다. 이 프롬프트의 경우에는, 프롬프트의 길이가 얼마나 되는지 상관 없이 동시에 표시될 것이라는 것을 의미합니다. backslash('\')에 의해 $()의 내용이 쉘 해석으로 부터 보호되어, "date" 명령이 새로운 프롬프트 가 생성될 때 마다 수행됩니다.

리눅스에는 많은 종류의, date, grep, 또는 wc 와 같은 조그마한 유틸리티 프로그램들이 부수되어 있어, 데이타를 처리할 수 있도록 합니다. 만약, 이들 프로그램들의 아주 복잡한 조합을 생성해서 프롬프트 안에 넣고자 하신 다면, 여러분 자신의 쉘 스크립트를 작성한 다음, 프롬프트에서 그것을 불러 들이는 방법이 더 쉬울 수 있읍니다. 적절한 때에 쉘 변수들이 풀어지도록 하기 위해 쉘 스크립트 내에서 자주 이스케잎 시퀀스들이 요구됩니다(위에서 보여진 날짜 명령어와 같이): 이것은 프롬프트 PS1 라인 내에서 한 단계 높은 수준으로 올라 가는 것이며, 쉘 스크립트를 생성하여 회피하는 방법은 아주 훌륭한 아이디어입니다.

프롬프트 내에서 사용되는 조그만 쉘 스크립트 예제가 아래에서 주어집니다.

```
#!/bin/bash
#     lsbytesum - sum the number of bytes in a directory listing
TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \n$TotalBytes/1048576 \nquit" | bc)
echo -n "$TotalMeg"
```

저는 이것을 때에 따라서 함수로(훨씬 효율적이지만-불행하게도 함수에 대한 자세한 설명은 이 문서의 범위 밖입니다), 또는 경로에 포함되어 있는 ' /bin' 디랙토리에 있는 쉘 스크립트로 보관했읍니다. 프롬프트에 사용되어 아래와 같은 결과가 됩니다.

```
[2158][giles@nikola:~]$ PS1="[\u@\h:\w (\$(lsbytesum) Mb)]\$ "
[giles@nikola:~ (0 Mb)]$ cd /bin
[giles@nikola:/bin (4.498 Mb)]$
```

3.3 프롬프트에서 필요한 그밖의 것들

대부분의 제가 만든 프롬프트에는 시각과 현재 디랙토리의 이름이 포함되어 있다는 것을 알 수 있읍니다. 여기에다가, 소형이지만 효과적인 Dan씨가 만든 프롬프트를 넣고자 합니다.

```
[giles@nikola:~]$ cur_tty=$(tty | sed -e "s/.*tty\(.*\)/\1/")
[giles@nikola:~]$ echo $cur_tty
p4
[giles@nikola:~]$ PS1="\!,$cur_tty,\$?\$ "
1095,p4,0$
```

디랙토리 경로를 바꿈에 따라서 프롬프트의 크기가 갑작스럽게 바뀔 수 있기 때문에 Dan씨는 프롬프트에 현재의 디랙토리가 표시되는 것을 좋아하지 않읍니다. 그래서 그는 디랙토리의 자취를 그의 머리속에 기억합니다(또는 "pwd"를 치기도 합니다.) Dan씨는 cash과 tcsh로 유닉스를 익혔기 때문에, 그는 명령어 history를 광범위 하게 사용하고 있으며(우리들의 대부분은 배쉬를 쓴다고 믿기에 그럴 필요가 없지만), 그래서, 그의 프롬프트의 첫번째 항목은 명령어 history 번호입니다. 두번째 항목은 tty의 중요 문자들 입니다("tty"의 출력을 'sed'로 모읍니다). 세번째 항은 마지막 command/pipeline의 exit 값입니다(note that this is rendered useless by any command executed within the prompt - you could work around that by capturing it to a variable and playing it back, though). 마지막으로, "\$"은 일반 사용자에게는 달러 기호를 나타내고, 사용자가 루트일 경우에는 해쉬 표시("#") 를 나타냅니다.

3.4 배쉬 환경과 함수들

앞에서 언급된 바와 같이, PS1, PS2, PS3, PS4 그리고 PROMPT_COMMAND 들은 모두 배쉬 환경으로 저장되어 있읍니다. DOS에 익숙한 분들은 아주 큰 덩어리의 정보를 환경으로 넘기는데에 대하여 두려운 생각을 가질 것입니다. 왜냐하면 DOS의 환경은 조그마했 으며, 잘 확장하지도 않았기 때문입니다. 환경에 넣을 수 있고 넣어야만 하는 것들에 대해서 실제적인 제약이 있을지도 모르지만, 그것들이 뭐가 될지도 모르고, 여기에서 는 DOS 사용자들에게 익숙한 그런 환경보다는 규모에 있어서 아마 몇배나 훨씬 더 큰 것에 대하여 이야기를 하고 있읍니다. 우리의 Dan씨가 말합니다:

"제가 사용하는 쉘에는, 62개의 알리아스와 25개의 함수가 있읍니다. 기계에서 직접 사용할 경우에, 어떤 것이 단독적으로 필요하고 bash에서 손으로 쉽게 씌어질 수 으면, 그것을 쉘 함수로 만든다는 것이 제일 첫번째 규칙입니다(알리아스로 쉽게 표현 할 수 없다는 가정하에서). 만약 메모리에 대해서 걱정하는 사람들이 있다면, 그들은 bash를 사용할 필요가 없읍니다. Bash는 저의 리눅서 기계에서 실행되는 가장 큰 프로그램 중의 하나입니다(Oracle을 제외하고). 때때로 'top'을 실행 시켜서 사용 메모리별로 분류하기 위해 'M'을 눌러 보세요-목록에서 bash가 얼마나 윗쪽에 가까이 있는지 살펴 보세요. 윽, sendmail보다도 더 덩치가 크군요! 그들 더러 가서 ash나 또는 다른 것들을 구하라고 하세요"

아마도 그는 그 작업을 하려고 하는 날에만 콘솔을 사용하지않나 추측합니다:X나 X응용 프로그램을 실행시키면, 배쉬보다 더 큰 것들이 많이 있읍니다. 그렇지만, 아이디어는 동일합니다:환경이라는 것은 사용되어야 하는 것이고, 그것이 차고 넘치는 것은 걱정할 필요가 없읍니다.

제가 이렇게 말할 때(너무 단순화 시켰다는 것에 대해서) 유닉스 구루들로 부터 책망 을 받을 각오를 하지만, 함수들이란 기본적으로 효율성을 위해 환경으로 적재되는 조그만 쉘 스크립트들입니다. Dan씨의 말을 다

시 인용합니다:"쉘 함수들은 가능한 만큼 효율화 한것입니다. It is the approximate equivalent of sourcing a bash/bourne shell script save that no file I/O need be done as the function is already in memory. 쉘 함수들은 전형적으로, 그것들이 최초의 쉘에서만 필요한지 또는 하부의 쉘에서도 요구되는가에 따라, [.bashrc 또는 .bash_profile]로 부터 적재됩니다. 쉘 스크립트 하나를 실행 시켜서 이것을 대비해 보십시오: 쉘이 분기하여, 자식 프로쎄스가 exec 를 수행하고, potentially 경로가 찾아지고, 커널이 파일을 열고 그 파일을 어떻게 실행할지 결정하기위해 충분한 바이트를 검사하며, 쉘 스크립트인 경우 쉘이 그 스크립트의 이름을 변수로 하여 시작하여야 하며, 그 다음에 파일을 열고, 읽고 그 문장을 수행합니다. 쉘 함수와 비교하여, 문장을 수행하는 것을 빼고는 다른 것들은 불필요한 것으로 간주됩니다."

제 4 절 X텀 제목 표시줄의 조작

Non-printing 이스케잎 시퀀스들을 사용하여 흥미있는 프롬프트를 생성할 수 있읍니 다. 이들 이스케잎 시퀀스를 사용하기 위해서는, 그것들을 \[와 \] 로 둘러 쌀 필요 가 있는데, 그렇게 함으로써 배쉬가 프롬프트의 길이를 계산할 때 그것들을 무시하도 록 할 수 있읍니다. 이들 구획 문자들을 빠뜨리게 되면 프롬프트의 실제 길이를 알 수 없게 되기 때문에, 커서가 엉뚱한 곳에 나타나게 됩니다. 또한, 이스케잎 시퀀스들의 앞에는 배쉬버전 2 이전의 경우에는 \033[, 그 이후의 버전인 경우에는 \033[이거나 \e[이 선행 되어야만 합니다.

콘솔에서, 프롬프트를 써서 X텀의 제목표시줄을 변경하고자 할 경우에는, 프롬프트에 잡동사니들이 나타나게 됩니다. 이것을 피하기 위해서, 프롬프트가 X텀에 있게 될 것이라는 것을 배쉬에게 일러 주기위해 TERM 환경변수를 테스트해 보십시오.

```
function proml
{
case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=''
        ;;
esac

PS1="${TITLEBAR}\
[\$(date +%H%M)]\
[\u@\h:\w]\
\$ "
PS2='> '
PS4='+ '
}
```

이것은 ' /.bashrc'에 포함될 수 있는 함수의 하나입니다. 함수의 이름을 불러서 실행할 수 있읍니다. PS1문자열과 마찬가지로, 함수도 환경에 저장되어 있읍니다. PS1문자열이 함수에 의해 한번 설정되면, 'unset proml' 명령으로 함수를 환경으로 부터 제거할 수 있읍니다. 프롬프트는 X텀에 있는 것에서 콘솔에 있는 것으로 변경할 수 없기 때문에, TERM 변수는 프롬프트가 생성될 때 마다 테스트 되지 않읍 니다. 프롬프트를 정의할 때 연속표시(backslashes'\')들을 사용하여, 여러 줄에 걸쳐 그것을 할 수 있읍니다. 이런 방법이 더 읽기 쉽게 해주며, 수정하거나 디버거 시에 유리합니다.

저는 이것을 함수로서 정의하는데, 배쉬프롬프트 패키지(이문서의 후반에서 논의함)가 프롬프트들을 다루는 방법이기 때문입니다:유일한 방법은 아니지만, 잘 작동합니다. 사용하는 프롬프트가 더 복잡해짐에 따라, 그것들을 프롬프트에 타이프하는 것이 더욱 더 성가시게 되므로, 그것들을 텍스트 파일에다 만들어 놓는 것이 더 실

용적입 니다. 이 경우에 있어서, 프롬프트에서 테스트하기 위해, "proml"이라는 텍스트 파일에 저장하십시오. 아래와 같이 작업을 할 수 있읍니다.

```
[giles@nikola:/bin (4.498 Mb)]$ cd          -> 프롬프트를 저장하고자 하는
                                               디랙토리로 이동
[giles@nikola:~ (0 Mb)]$ vi proml           -> 프롬프트 파일을 작성함
...                                         -> 위에 주어진 내용을 입력함
[giles@nikola:~ (0 Mb)]$ source proml       -> 프롬프트 함수를 읽어 들임
[giles@nikola:~ (0 Mb)]$ proml              -> 프롬프트 함수를 실행
```

이 프롬프트를 생성하는 데 있어서 첫번째 스텝은, 우리들이 시작하는 쉘이 X텀인지 아닌지를 테스트합니다:만약 그렇다면, 쉘 변수 (${TITLEBAR})가 정의됩니다. 그것은 적당한 이스케잎 시퀀스들과 X텀 제목표시줄에 <사용자>@<기계>:<작업 디랙 토리>를 표시하는 \u@\h:\w 로 구성됩니다. 이것은 특별히 최소화된 X텀들에서 유용하고, 더욱 신속하게 터미널을 확인하도록 해 줍니다. 이 프롬프트에 있는 다른 것들은 우리가 이전에 만들었던 것들과 유사할 것입니다.

이런 식으로 X텀 제목표시줄을 조작하는 데 있어 단 하나의 약점은, 제목표시줄 hack를 설정하지 않은 시스템으로 로그인해 들어 갈 때 나타납니다:X텀은 제목 표시줄 hack 가 위치하는 이전 시스템으로 부터의 정보를 계속해서 보여 줄 것입니다.

제 5 절 색

앞에서 언급한 바와 같이, non-printing 이스케잎 시퀀스들은 \[\033[와 \]로 둘러 싸여 있어야만 합니다. 색 이스케잎 시퀀스들의 경우에는, 소문자 m도 같이 넣어야 합니다.

아래와 같은 프롬프트들을 시도해서, 명명된 색깔들이 나타나지 않는다면, ' /.Xdefaults' 파일(혹은 같은 종류의 파일)을 검토해서 "XTerm*Foreground: BlanchedAlmond" 와 같은 줄이 있는지 살펴 보십시오. 그 줄의 앞쪽에 느낌표('!')를 삽입해서 주석 처리할 수 있읍니다. 물론, 이 작업은 사용하는 터미널 에뮬레이터의 종류에 따라 달라 집니다. 이것이 터미널의 전경 색깔을 무효화 할 수 있는 가장 가능성이 있는 곳입니다.

프롬프트에 파란색 텍스트를 포함하기 위해 :

PS1="\[\033[34m\][\$(date %H%M)][\u@\h:\w]$ "

이 프롬프트의 문제점은, 34번 색 코드로 시작하는 파란색깔이 통상적인 회색으로 되돌아 가지 않아서, 프롬프트 뒤에 써 넣는 어떤 종류의 텍스트라도 프롬프트와 같은 색을 가지게 된다는 것입니다. 이것은 어두운 파란색이라서, 강조 코드가 약간은 도움이 될 수 있읍니다.

PS1="\[\033[1;34m\][\$(date %H%M)][\u@\h:\w]$\[\033[0;37m\] "

이제는 밝은 파란색으로 되며, 대부분의 사람들이 바라는 회색으로 되돌아 가서 끝 나게 됩니다.

주의 : 다음과 같은 점을 지적해 두고자 합니다. 만약, X텀을 사용하면서, 터미널의 색깔에 대해 "xterm -bg midnightblue -fg yellow" 과 같은식으로 설정해 놓으셨다면, 위에서 기술한 바와 같이 색깔을 되돌리는 작업이 여러분들이 설정한 "-fg"을 무효화 시킨다는 것입니다. 저는 이문제를 해결하지 못했으며, 여러분들이 해법을 발견하게 되면 저에게 일러 주시기 바랍니다. 여러분들이 밝은 색의 배경을 사용하고 계신다면, 위에서 사용된 회색 텍스트는 읽기가 곤란할 것입니다. 또한, X텀에서 밝은 배경을 사용하면서, 콘솔과 X텀 양쪽에서 프롬프트를 사용하기를 원하시면 제목 표시줄 절에 서 보인 바와 같이 "case $TERM in ..." 로써 색 설정에 대해서 약간의 조정을 할 필요가 있읍니다.

아래에 색상표가 주어집니다:

```
Black       0;30     Dark Gray     1;30
Blue        0;34     Light Blue    1;34
Green       0;32     Light Green   1;32
```

```
Cyan        0;36     Light Cyan    1;36
Red         0;31     Light Red     1;31
Purple      0;35     Light Purple  1;35
Brown       0;33     Yellow        1;33
Light Gray  0;37     White         1;37
```

배경색도 또한 설정할 수 있는데, 파란 배경에 대해서는 44, 빨간색 배경에 대해서는 41, 등을 사용할 수가 있읍니다. 배경색에 대해서는 강조 색은 없읍니다. 파란색 배경에 밝은 빨간색 텍스트와 같이 조합을 해서도 사용할 수있읍니다: \[\033[44;1;31m\], 그렇지만, 따로 따로 색깔을 설정하는 것이 더 잘 작동하는 것 같읍니다(즉, \[\033[44m\]\[\033[1;31m\]). 다른 코드들도 포함하여 사용할 수 있는데, 4:밑줄, 5:깜박거림, 7:반전 그리고 8:숨김 등이 있읍니다.

프롬프트 패키지(제가 수정을 하여, 원문을 보기위해 요구되는 특별한 X텀 폰트들 을 사용하기 보다는 표준적인 콘솔에서 더 잘 작동하도록 만들었읍니다)에 있는 "elite2"라고 이름 붙인 프롬프트에 근거하여, 아래의 프롬프트는 제가 아주 많이 사용하는 것입니다.

```
function elite
{

local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local CYAN="\[\033[0;36m\]"
local LIGHT_CYAN="\[\033[1;36m\]"

case $TERM in
    xterm*)
        local TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        local TITLEBAR=""
        ;;
esac

local GRAD1=$(tty|cut -d/ -f3)
PS1="$TITLEBAR\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\u$GRAY@$CYAN\h\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\#$GRAY/$CYAN$GRAD1\
$LIGHT_CYAN)$CYAN-$LIGHT_CYAN(\
$CYAN\$(date %H%M)$GRAY/$CYAN\$(date %d-%b-%y)\
$LIGHT_CYAN)$CYAN-$GRAY-\
$LIGHT_GRAY\n\
$GRAY-$CYAN-$LIGHT_CYAN(\
$CYAN\$$GRAY:$CYAN\w\
$LIGHT_CYAN)$CYAN-$GRAY-$LIGHT_GRAY "
PS2="$LIGHT_CYAN-$CYAN-$GRAY-$LIGHT_GRAY "
}
```

색깔들을 읽기 쉬운 이름으로 임시 쉘 변수로 정의 하였읍니다. 이렇게 하면, 작업 하기에 편리합니다. "GRAD1"변수가 사용되어, 사용하는 터미널의 종류를 결정합니다. X텀에서 작업을 하는지 결정하기 위해

테스트한 것과 마찬가지로, 단지 한번만 실행될 필요가 있읍니다. 색깔을 제외하면, 아래와 같은 프롬프트를 보시게 됩니다.

```
--(giles@nikola)-(75/ttyp7)-(1908/12-Oct-98)--
--($:~/tmp)--
```

제 6 절 배쉬 프롬프트 패키지

6.1 입수 가능한 곳

배쉬 프롬프트 패키지는 <http://bash.current.nu> 에서 구할 수있으며, 이 작업은 Rob Current(aka BadLandZ) 씨의 조정하에 몇몇 사람들이 참여하여 이루 어진 것입니다. 이 패키지는 예전 부터 베타 버전이지만, 다중 프롬프트(theme)를 사용할 수 있도록 하는 아주 간단한 방법을 제공하여, 로그인 쉘과 하부쉘에서 프롬프트를 설정할 수 있게 합니다(즉, PS1 문자열을 ' /.bash_profile'과 ' /.bashrc' 파일에 저장 함으로써). 대부분의 테마(theme)들은 확장 VGA 문자 세트들을 사용하기 때문에, VGA 폰트(대부분의 시스템에서 디폴트가 아님)에 익숙하지 않으신 분들에게는 좋지 않게 보입니다.

6.2 X텀 폰트 바꾸기

배쉬 프롬프트 패키지에 있는 가장 매력적인 프롬프트 몇개를 사용히기 위해서는, 프롬프트에 쓰일 문자세트들을 지원하는 폰트를 구해서 설치할 필요가 있읍니다. 이것들은 "VGA 폰트"라고 하지만, 리눅스에 일반적으로 적재되는 폰트들-확실히 다른 문자 세트를 지원하기는 하지만-과의 차이가 명확하지는 않다고 봅니다. 표준 X텀 폰트들은, 많은 종류의 강조 문자들을 포함해서, 확장 알파베트를 지원합니다. VGA 폰트에서는, 이러한 것들은 그래픽 문자들-블록, 점, 선들-로 대치됩니다. 이것을 더 자세히 설명할 수 있는 분이 계시다면, e-메일을 보내 주시기 바라며, 여기에 포함 시키도록 하겠읍니다.

이들 폰트를 구해서 설치하는 것은 약간 복잡한 과정을 거칩니다. 우선, 그 폰트(들) 을 복구합니다. 다음으로, 그것들이 .pcf 또는 .pcf.gz 형식의 파일인지 확인합니다. 만약, 그것들이 .bdf 파일이면, "bdftopcf"명령을 조사합니다(즉, 맨 페이지를 읽어 봅니다). 그리고 .pcf 또는 .pcf.gz 파일들을 '/usr/X11R6/lib/X11/fonts/misc' (RedHat5.1 과 Slackware3.4 의 경우에는 이 디랙토리가 올바른 디랙토리이지만, 그 밖의 배포본에서는 다른 디랙토리가 될 수도 있읍니다) 디랙토리에 옮기세요. "cd" 명령으로 그 디랙토리로 바꾼 다음에, "mkfontdir" 명령을 실행 시킵니다. 그 다음에 "xset fp rehash" 를 실행합니다. 때때로, 같은 디랙토리에 있는 'fonts.alias' 파일 을 편집해서, 그 폰트에 해당하는 더 짧은 알리아스 이름을 만드는 것도 훌륭한 아이디어입니다.

새로운 폰트들을 사용하기 위해서는, 여러분의 X텀에 맞는 적당한 명령어로써 선택된 옵션들을 포함해서 X텀 프로그램을 시작하면 되는데, 이러한 작업에 대해서는 맨 페이지나 또는 명령어 줄에서 "–help"를 사용해서 알아 낼 수 있읍니다. 아래의 예는 대중적으로 많이 쓰이는 용어 입니다:

```
xterm -font <fontname>
```

또는

```
xrerm -fn <fontname> -fb <fontname-bold>
Eterm -f <fontname>
rxvt -fn <fontname>
```

VGA 폰트들은 <http://hoome.earthlink.net/~us5zahns/enl/ansifont.htm> 의 'Stumpy's ANSI Fonts' 페이지에서 구할 수 있읍니다(제가 이 글을 쓰면서 많은 부분을 참조한 페이지임).

제 7 절 다른 프롬프트 불러 들이기

7.1 조금 후에, 다른 프롬프트 불러 들이기

이 HOWTO 문서에서, PS1 환경 변수들을 어떻게 만들며, 또 이들 PS1과 PS2 문자열들을 어떻게 함수에 합병해 넣어서, ' /.bashrc' 나 프롬프트 페키지의 테마로써 실행될 수 있는 지에 대하여 설명하였읍니다.

배쉬 프롬프트 페키지를 사용할 경우에, 'bashprompt -i'라고 입력하여 사용 가능한 테마들의 목록을 볼 수 있읍니다. 앞으로의 로그인 쉘(여러분들의 X텀 설정에 따라 다르기는 하지만, 주로 콘솔, 뿐만 아니라, 텔넷이나 X텀)에서 프롬프트를 설정하기 위해서는, 'bashprompt -l themename'을 입력하면 됩니다. 그러면, 요청된 테마가 시작될 때, 그것을 부를 수 있도록, 배쉬프롬프트는 ' /.bash_profile'을 수정하게 됩니다. 앞으로의 하부 쉘(대개, X텀, rxvt, 등)에서의 프롬프트를 설정하기위해서는, 'bashprompt -s themename'을 입력하면 되며, 배쉬프롬프트는 ' /.bash' 파일을 수정하여, 기동할 때, 적절한 테마를 부를 수 있도록 합니다.

7.2 지금 즉시, 다른 프롬프트 불러 들이기

현재의 터미널에서(위에서 제시된 예인 'elite'함수를 사용해서) 프롬프트를 바꿀 수 있는데, 'source elite'라고 친 다음에 'elite'('elite'함수 파일이 작업 디렉토리에 있다고 가정)를 입력하면 됩니다. 이렇게 하는 것은 약간 성가신 작업이며, 여러분의 환경 공간에 여분의 함수(elite)를 남겨 놓는 것이 됩니다-환경에서 그것을 깨끗이 지우고 싶으시면, 역시, 'unset elite'라고 입력하시면 됩니다. 이렇게 하는 것이 조그만 쉘 스크립트에 대해서는 이상적인 방법 같이 보이지만, 여기에서는 스크립트는 작동하지 않읍니다. 왜냐하면, 스크립트로는 현재 쉘의 환경을 변경할 수 없기 때문입니다:스크립트는 하부 쉘의 환경만 바꿀 수 있읍니다. 스크립트가 정지하는 순간에, 하부 쉘은 사라지게 되고, 그 스크립트에 의해서 변경 된 하부 쉘의 환경도 사라지게 됩니다. 현재 쉘의 환경 변수들을 바꿀 수 있는 것은 환경 함수들 뿐입니다. 배쉬프롬프트 페키지에 의해서, "callbashprompt"라고 불리는 함수가 환경에 추가되며, 그리고, while they don't document it, 배쉬 진행 중에, 그 함수가 불리어져, 어떤 배쉬 프롬프트라도 적재 할 수 있읍니다. 그 함수는 테마가 설치된 디렉토리(여러분이 부를려고 하는 테마는 여기에 있어야 합니다)를 살펴 보고, 요청된 함수를 소스화 해서, 그 함수를 적재하고, 그리고나서 그함수를 내리게 되어, 환경을 정돈된 상태로 유지합니다. "callbashprompt"는 이렇게 사용되기로 의도된 것도 아니며, 오류 검사도 없지만, 이 점만 명심한다면, 잘 작동합니다.

제 8 절 프롬프트 색을 동적으로 불러 들이기

8.1 예 "Proof of Concept"

아래의 예는 흥미로운 프롬프트라기 보다는 "proof of concept" 입니다:동적으로 프롬프트 안의 색깔을 바꾸는 것입니다. 이 예에서는, 시스템의 부하에 따라서, 호스트 이름의 색깔을 바꾸게 됩니다(경고 로서).

```
#!/bin/bash
#   "hostloadcolour" - 17 October 98, by Giles
#
#   The idea here is to change the colour of the host name in the prompt,
#   depending on a threshold load value.

# THRESHOLD_LOAD is the value of the one minute load (multiplied
# by one hundred) at which you want
# the prompt to change from COLOUR_LOW to COLOUR_HIGH
THRESHOLD_LOAD=200
COLOUR_LOW='1;34'
          # light blue
```

```
COLOUR_HIGH='1;31'
           # light red

function prompt_command {
ONE=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
#   Apparently, "scale" in bc doesn't apply to multiplication, but does
#   apply to division.
ONEHUNDRED=$(echo -e "scale=0 \n $ONE/0.01 \nquit \n" | bc)
if [ $ONEHUNDRED -gt $THRESHOLD_LOAD ]
then
    HOST_COLOUR=$COLOUR_HIGH
        # Light Red
else
    HOST_COLOUR=$COLOUR_LOW
        # Light Blue
fi
}

function hostloadcolour {

PROMPT_COMMAND=prompt_command
PS1="[$(date +%H%M)][\u@\[\033[\$(echo -n \$HOST_COLOUR)m\]\h\[\033[0;37m\]:\w]$ "
}
```

여러분이 선호하시는 편집기를 이용해서, 이 파일을 "hostloadcolour"라는 이름으로 저장하십시오. 만약, 배쉬프롬프트 패키지를 이미 설치하셨다면, 이 파일은 테마의 하나로서 작동할 것입니다. 배쉬프롬프트를 설치하지 않았다면, 'source hostloadcolour'라고 치고나서, 'hostloadcolour'를 입력하십시오. 어느 쪽이든지, "prompt_command"는 여러분들의 환경에서 함수들 중 하나가 됩니다. 위 코드를 잘 살펴 보시면, 색깔들($COLOUR_HIGH 와 $COLOUR_LOW)이 부분적인 색상 코드(즉, "\[\033[1;34m\]" 대신에 "1;34")로 만 설정되어 있다는 것을 아시게 될 것입니다. 이 방법을 저는 더 선호합니다. 저는 완성된 코드로 작동하게 할 수가 없었는데, 이 문제를 다룰 수 있으면 저에게 일러 주십시오.

제 9 절 예제 프롬프트들

9.1 "Lightweight" 프롬프트

```
function proml {
local BLUE="\[\033[0;34m\]"
local RED="\[\033[0;31m\]"
local LIGHT_RED="\[\033[1;31m\]"
local WHITE="\[\033[1;37m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
```

```
esac

PS1="${TITLEBAR}\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$LIGHT_RED\u@\h:\w$BLUE]\
$WHITE\$$LIGHT_GRAY "
PS2='> '
PS4='+ '
}
```

9.2 배쉬프롬프트 테마에서의 'elite'

이 프롬프트는 VGA 폰트가 필요하다는 점에 유의 하십시오.

```
# Created by KrON from windowmaker on IRC
# Changed by Spidey 08/06
function elite {
PS1="\[\033[31m\]\332\304\[\033[34m\](\[\033[31m\]\u\[\033[34m\]@\[\033[31m\]\h\
\[\033[34m\])\[\033[31m\]-\[\033[34m\](\[\033[31m\]\$(date +%I:%M%P)\
\[\033[34m\]-:-\[\033[31m\]\$(date +%m)\[\033[34m\033[31m\]/\$(date +%d)\
\[\033[34m\])\[\033[31m\]\304-\[\033[34m]\\371\[\033[31m\]-\371\371\
\[\033[34m\]\372\n\[\033[31m\]\300\304\[\033[34m\](\[\033[31m\]\W\[\033[34m\])\
\[\033[31m\]\304\371\[\033[34m\]\372\[\033[00m\]"
PS2="> "
}
```

9.3 "Power User" 프롬프트

저는 실제로 이 프롬프트를 사용하고 있지만, PII 400, single-user 시스템에서 프롬프트가 나타나는 데는 감지할 수있을 정도의 지연이 나타나므로, PII 100 또는 다른 muti-user 시스템에는 사용하기를 권장하지 않습니다. 실제적인 프롬프트로 쓰기 위해서 보다는, 아이디어를 얻기 위해 잘 살펴 보시기 바랍니다.

```
#!/bin/bash
#----------------------------------------------------------------------
#       POWER USER PROMPT "pprom2"
#----------------------------------------------------------------------
#
#   Created August 98, Last Modified 9 November 98 by Giles
#
#   Problem: when load is going down, it says "1.35down-.08", get rid
#   of the negative
#   - limit the length of the $PWD to 30 chars - right-truncate if
#   it exceeds that

function prompt_command
{
#   Create TotalMeg variable: sum of visible file sizes in current directory
local TotalBytes=0
for Bytes in $(ls -l | grep "^-" | cut -c30-41)
do
```

```
    let TotalBytes=$TotalBytes+$Bytes
done
TotalMeg=$(echo -e "scale=3 \nx=$TotalBytes/1048576\n if (x<1) {print \"0\"} \n print x \nquit" | bc)

#       This is used to calculate the differential in load values
#       provided by the "uptime" command.  "uptime" gives load
#       averages at 1, 5, and 15 minute marks.
#
local one=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\)/\1/" -e "s/ //g")
local five=$(uptime | sed -e "s/.*load average: \(.*\...\), \(.*\...\), \(.*\...\).*/\2/" -e "s/ //g")
local diff1_5=$(echo -e "scale = scale ($one) \nx=$one - $five\n if (x>0) {print \"up\"} else {print \"down\"}\n print x \nquit \n" | bc)
loaddiff="$(echo -n "${one}${diff1_5}")"

#   Count visible files:
let files=$(ls -l | grep "^-" | wc -l | tr -d " ")
let hiddenfiles=$(ls -l -d .* | grep "^-" | wc -l | tr -d " ")
let executables=$(ls -l | grep ^-..x | wc -l | tr -d " ")
let directories=$(ls -l | grep "^d" | wc -l | tr -d " ")
let hiddendirectories=$(ls -l -d .* | grep "^d" | wc -l | tr -d " ")-2
let linktemp=$(ls -l | grep "^l" | wc -l | tr -d " ")
if [ "$linktemp" -eq "0" ]
then
    links=""
else
    links=" ${linktemp}l"
fi
unset linktemp
let devicetemp=$(ls -l | grep "^[bc]" | wc -l | tr -d " ")
if [ "$devicetemp" -eq "0" ]
then
    devices=""
else
    devices=" ${devicetemp}bc"
fi
unset devicetemp

}

PROMPT_COMMAND=prompt_command

function pprom2 {

local        BLUE="\[\033[0;34m\]"
local  LIGHT_GRAY="\[\033[0;37m\]"
local LIGHT_GREEN="\[\033[1;32m\]"
local  LIGHT_BLUE="\[\033[1;34m\]"
local  LIGHT_CYAN="\[\033[1;36m\]"
local      YELLOW="\[\033[1;33m\]"
local       WHITE="\[\033[1;37m\]"
local         RED="\[\033[0;31m\]"
```

```
case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$BLUE[$RED\$(date +%H%M)$BLUE]\
$BLUE[$RED\u@\h$BLUE]\
$BLUE[\
$LIGHT_GRAY\${files}.\${hiddenfiles}-\
$LIGHT_GREEN\${executables}x \
$LIGHT_GRAY(\${TotalMeg}Mb) \
$LIGHT_BLUE\${directories}.\
\${hiddendirectories}d\
$LIGHT_CYAN\${links}\
$YELLOW\${devices}\
$BLUE]\
$BLUE[${WHITE}\${loaddiff}$BLUE]\
$BLUE[\
$WHITE\$(ps ax | wc -l | sed -e \"s: ::g\")proc\
$BLUE]\
\n\
$BLUE[$RED\$PWD$BLUE]\
$WHITE\$\
\
$LIGHT_GRAY "
PS2='> '
PS4='+ '
}
```

9.4 터미널의 폭 프롬프트

한 친구가 불평을 했는데, 프롬프트 안에 $PWD 명령이 있어서 계속 길이가 바뀌는 그런 프롬프트를 좋아 하지않는다고 했읍니다. 그래서, 여러분들의 터미널의 폭에 꼭 맞도록 길이를 조정할 수 있는 프롬프트를 작성 하였읍니다.

```
#!/bin/bash

#   termwide prompt
#      by Giles - created 2 November 98
#                 last modified 9 November 98
#
#   The idea here is to have the upper line of this two line prompt
#   always be the width of your term.  Do this by calculating the
#   width of the text elements, and putting in fill as appropriate
```

```
#   or right-truncating $PWD.
#
#   - needs a lot of refinement, but it works
#   - host doesn't need to be figured out every time: that's a
#     constant
#   - what about username?  could change?
#   - text vars generated by prompt_command can't be locals,
#     because they have to go to PS1.

function prompt_command {

TERMWIDTH=${COLUMNS}

#   Calculate the width of the prompt:

hostnam=$(echo -n $HOSTNAME | sed -e "s/[\.].*//")
let hostsize=$(echo -n $hostnam | wc -c | tr -d " ")
#   "whoami" and "pwd" include a trailing newline
usernam=$(whoami)
let usersize=$(echo -n $usernam | wc -c | tr -d " ")
newPWD="${PWD}"
let pwdsize=$(echo -n ${newPWD} | wc -c | tr -d " ")
#   Add all the accessories below ...
let promptsize=$(echo -n "--(${usernam}@${hostnam})---(${PWD})--" \
                 | wc -c | tr -d " ")
let fillsize=${TERMWIDTH}-${promptsize}
fill=""
while [ "$fillsize" -gt "0" ]
do
    fill="${fill}-"
        let fillsize=${fillsize}-1
done

if [ "$fillsize" -lt "0" ]
then
    let cut=((${fillsize})*(-1))+3
    sedvar=""
    while [ "$cut" -gt "0" ]
    do
        sedvar="${sedvar}."
    let cut=${cut}-1
    done
    newPWD="...$(echo -n $PWD | sed -e "s/\(^${sedvar}\)\(.*\)/\2/")"
fi
}

PROMPT_COMMAND=prompt_command

function termwide {
```

```
local GRAY="\[\033[1;30m\]"
local LIGHT_GRAY="\[\033[0;37m\]"
local WHITE="\[\033[1;37m\]"

local LIGHT_BLUE="\[\033[1;34m\]"
local YELLOW="\[\033[1;33m\]"

case $TERM in
    xterm*)
        TITLEBAR='\[\033]0;\u@\h:\w\007\]'
        ;;
    *)
        TITLEBAR=""
        ;;
esac

PS1="$TITLEBAR\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\${usernam}$LIGHT_BLUE@$YELLOW\${hostnam}\
${LIGHT_BLUE})-${YELLOW}-\${fill}${LIGHT_BLUE}-(\
$YELLOW\${newPWD}\
$LIGHT_BLUE)-$YELLOW-\
\n\
$YELLOW-$LIGHT_BLUE-(\
$YELLOW\$(date +%H%M)$LIGHT_BLUE:$YELLOW\$(date \"+%a,%d %b %y\")\
$LIGHT_BLUE:$WHITE\$$LIGHT_BLUE)-\
$YELLOW-\
$LIGHT_GRAY "

PS2="$LIGHT_BLUE-$YELLOW-$YELLOW-$LIGHT_GRAY "
}
```