July 28, 2012
Posted by: gaHcep
Category: Coding, Linux, Bash

Препарируем Bash Command Shell Prompt

Сколько времени вы проводите за набором команд в юниксовом терминале, шелле? Если так же много как я (а это каждый рабочий, да и не только, день), то наверняка задумывались о том, как же повысить ее информативность, поменять цветовой и смысловой вывод. Сделать это можно и даже нужно. Сегодня я расскажу о переменных окружения ***Bash Shell Prompt*** и для чего они нужны; мы узнаем, что такое терминал, что значат escape последовательности, как правильно их нужно применять, узнаем подробнее о *PS переменных* и научимся добавлять требуемый функционал в .bashrc файлы. Из всего многообразия escape кодов для терминалов, мы поработаем детально с двумя крупнейшими группами – группа кодов для преобразования внешнего вида текста и группа кодов для позиционирования курсора.

VT100, terminfo, xterm

Начнем с того, что **bash**, равно как и **zsh** и другие шеллы
являются **эмуляторами** терминала. Наследие можно проследить от широко известного терминала DEC VT100. **VT100** согласно Википедии [1]
… стал для DEC первым массово производимым терминалом, широко использующим графические возможности (мерцание, полужирный шрифт, инвертирование цветов, подчёркивание) и позволяющий выбрать режим отображения 80 или 132 символов в строке. Все настройки VT100 производились при помощи интерактивных диалогов появлявшихся на экране и сохранялись в энергонезависимой памяти, встроенной в сам терминал. В VT100 также имел дополнительный набор псевдографических символов
Что самое главное, терминал управлялся набором команд, называемыми также **escape кодами ANSI**. И хоть после VT100 были разработаны более продвинутые терминалы, например серии VT200, но набор команд VT100 стал де-фактом для современных эмуляторов. Поэтому далее в тексте будет приниматься за дефолтный тип терминала именно VT100 и отсылка на более детальное описание команд будет идти на ресурсы, где описываются ANSI escape коды именно терминала VT100 как наиболее общеприменимые.
xterm – данным термином называют эмулятор терминала в среде X Window System. Согласно той же Википедии [2]:
xterm была разработана как самостоятельная программа – эмулятор терминала для VAXStation 100 (VX100)
компании DEC. Со временем **xterm** стал частью X, хоть изначально это и не планировалось. Теперешние шеллы начинали разрабатываться как варианты xterm (в современном терминале есть переменная **$TERM**, значение которой в bash, zsh как раз и равно “xterm”). Кстати SSH клиент putty также эмулирует xterm.
terminfo (бывший termcap) — база данных описывающая возможности того или иного терминала.
Может показаться, что все шеллы поддерживают строго один тип терминала. Это не так. Какой-нибудь эмулятор может поддерживать **vt220** и наследовать другое подмножество команд предоставляемое пользователю. Кстати, в большинстве стартовых скриптов относящихся к работе терминала (шелла), нередка проверка на тип переменной **$TERM**.

Для получения дополнительной информации советую прочесть больше информации о том терминале, который вы используете. Для получения справочной информации можно воспользоваться командой “**man terminfo**”.

Переменные окружения

Прежде чем окончательно приступить к главной теме статьи, замечу, что все тесты и настройка переменных проводилась в CentOS 6.1 и для других систем (MacOS, к примеру) написанное может отличаться. Примеры тестировались на bash 4.2.1.

Работа с командной строкой помимо творческого может доставлять и эстетическое удовольствие при использовании (одного знания об их существовании обычно не хватает) нескольких переменных командной строки.

К примеру, вот так выглядит по-умолчанию строка ввода в шелле в CentOS:

```
[gahcep@localhost ~]$
```

Почему именно так? Скоро узнаем. А пока,
встречаем: **PS1**, **PS2**, **PS3**, **PS4** и **PROMPT_COMMAND**. Именно **PS1** в ответе за внешний вид строки ввода и в основном именно ему и будет посвящен пост. И да, **PS** от **P**rompt **S**tatement.

PS1 (Default interaction prompt)

Командная строка ввода. Именно она отвечает за то, что предшествует текущему символу и той команде, которая в данный момент печатается. Пример выше видели? А вот значение PS1 для него:

```
[gahcep@localhost ~]$ echo  $PS1
[\u@\h  \W]\$
```

Вот это и есть содержимое PS1 (закодированное с помощью нескольких символьных последовательностей): открывающая скобка ‘**[**’, затем имя пользователя (**u**sername) [**\u**], затем символ ‘**@**’, потом имя хоста (**h**ostname) [**\h**], текущая директория (**w**orkdir) [**\W**], закрывающая скобка ‘**]**’ и в конце символ ‘**$**’ предваряющий командный ввод. Более подробно о PS1 – в следующем разделе.

PS2 (Continuation interactive prompt)

Часто при вводе очень длинной команды (или аргументов) для удобства принято разбивать ее (обычно символом ‘*****’) на несколько строк. То, что будет отображено при каждом вводе новой строки, являющейся продолжением предыдущей, содержится в этой переменной окружения. Значение по-умолчанию:

```
[gahcep@localhost ~]$ echo  $PS2
>
```

Пример ввода (не судить строго, пример синтетический):

```
[gahcep@localhost ~]$ ls --almost-all --author --ignore-backups \
> --color=auto --format=l --group-directories-first \
> --human-readable --indicator-style='file-type'
```

Как видно, символ ‘>’ простой. Попробуем сменить на “**next–>** ” (в конце пробел):

```
[gahcep@localhost ~]$ export PS2="next-> "
```

```
[gahcep@localhost ~]$ ls --almost-all --author --ignore-backups \
next-> --color=auto --format=l --group-directories-first \
next-> --human-readable --indicator-style='file-type'
```

Удобнее, правда? Хотя кому как...

PS3 (Select related interactive prompt)

Данная строка ввода используется в операторе select shell скрипта. По умолчанию переменная пуста:

```
[gahcep@localhost ~]$ echo  $PS3
```

Вообще это можно объяснить тем, что значение переменной PS3 имеет смысл лишь в контексте выполнения какой либо программы или скрипта. Давайте рассмотрим в качестве примера небольшой скрипт, печатающий сезон к которому относится выбранный пользователем месяц. Скрипт:

```
PS3="Choose a month: "
select m in Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec quit
do
case $m in
Dec|Jan|Feb) echo $m "is a winter month";;
Mar|Apr|May) echo $m "is a spring month";;
Jun|Jul|Aug) echo $m "is a summer month";;
Sep|Oct|Nov) echo $m "is an autumn month";;
quit) exit;;
esac
done
```

Как видите, значение переменной PS3 задается в первой строке:

```
PS3="Choose a month: "
```

и оно используется лишь внутри скрипта. Вот как выглядит работа со скриптом:

```
[gahcep@localhost ~]$ ./month-select.sh
1) Jan 3) Mar 5) May 7) Jul 9) Sep 11) Nov 13) quit
2) Feb 4) Apr 6) Jun 8) Aug 10) Oct 12) Dec
Choose a month: 2
Feb is a winter month
Choose a month: 7
Jul is a summer month
Choose a month: 13
[gahcep@localhost ~]$
```

Удобно и по делу. А вот как могло бы быть без использования PS3:

```
[gahcep@localhost ~]$ ./month-select.sh
1) Jan 3) Mar 5) May 7) Jul 9) Sep 11) Nov 13) quit
2) Feb 4) Apr 6) Jun 8) Aug 10) Oct 12) Dec
#? 2
Feb is a winter month
#? 7
Jul is a summer month
#? 13
```

```
[gahcep@localhost ~]$
```

Совсем не информативно.

PS4 (Trace related interactive prompt)

Эта переменная определяет внешний вид строки ввода, который появляется при отладке скриптов. Рассмотрим пример (файл **ps4-demo.sh**):

```
set -x
echo "Test PS4 script file"
ls -la /bin | grep "name" | wc -l
df -h /etc
ps -ely | grep sh | wc -c
```

Вывод:

```
[gahcep@localhost ~]$ ./ps4-demo.sh
++ echo 'Test PS4 script file'
Test PS4 script file
++ wc -l
++ grep name
++ ls -la /bin
7
++ df -h /etc
Filesystem Size Used Avail Use% Mounted on
/dev/sda2 7.7G 3.7G 3.6G 51% /
++ wc -c
++ grep sh
++ ps -ely
895
```

Два плюса что-то совсем не информативно, не так ли? Предлагаю добавить вывод даты и времени, имени файла и номера строки:

- **date +%D.%T.%-3N** – формат даты и времени. Более подробнее о синтаксисе прошу вызвать команду man date, а лучше info coreutils ‘date invocation’
- **date +%D.%T.%-3N** – формат даты и времени. Более подробнее о синтаксисе прошу вызвать команду man date, а лучше info coreutils ‘date invocation’
- **$LINENO** – номер текущей строки

Теперь в скрипте добавилась команда задания значения PS4. Смотрим:

```
PS4=' $(date +\%D.\%T.\%-3N)::`basename $0` [$LINENO]: '
set -x
echo "Test PS4 script file"
ls -la /bin | grep "name" | wc -l
df -h /etc
ps -ely | grep sh | wc -c
```

Теперь вывод:

```
[gahcep@localhost ~]$ ./ps4-demo.sh
02/09/12.05:42:45.130::ps4-demo.sh [4]: echo 'Test PS4 script file'
Test PS4 script file
02/09/12.05:42:45.140::ps4-demo.sh [5]: wc -l
```

```
02/09/12.05:42:45.138::ps4-demo.sh [5]: grep name
02/09/12.05:42:45.140::ps4-demo.sh [5]: ls -la /bin
7
02/09/12.05:42:45.160::ps4-demo.sh [6]: df -h /etc
Filesystem Size Used Avail Use% Mounted on
/dev/sda2 7.7G 3.7G 3.6G 51% /
02/09/12.05:42:45.166::ps4-demo.sh [7]: grep sh
02/09/12.05:42:45.165::ps4-demo.sh [7]: wc -c
02/09/12.05:42:45.167::ps4-demo.sh [7]: ps -ely
895
```

На порядок информативнее. Теперь можно профилировать выполнение конкретных функций или команд. Вообще для целей дебага bash скриптов существует целый ряд переменных **BASH*** (таких как **BASHPID**, **BASH_ARG**, **BASH_LINENO** – подробнее о них в bash мануале в разделе "**Shell Variables**") или в [3].

PROMPT_COMMAND

Bash выполняет то что написано в данной переменной (это может быть одна команда, а может быть несколько) до вывода PS1. Например, вот так можно заставить shell выводить время перед каждой новой командой:

```
[gahcep@localhost ~]$ export PROMPT_COMMAND="date +\%D::\%m:\%S"
02/09/12::02:24
[gahcep@localhost ~]$
```

Если же вывод должен быть в той же строке что и PS1, надо к команде добавить "**echo –n**" (**–n** позволит не учитывать завершающий символ перевода строки):

```
[gahcep@localhost ~]$ export PROMPT_COMMAND="echo -n [$(date +\%D::\%m:\%S)]"
[02/10/12::02:31][gahcep@localhost ~]$
```

Во всех примерах для изменения переменных, мы использовали ключевое слово **export**. Но это гарантирует изменения лишь на *текущий* сеанс. Чтобы изменения вступили в действие на *постоянной*основе, следует добавить эти команды (**export …**) в **.bashrc** и/или **.bash_profile**, файлы, расположенные, как правило, в домашней директории.

Файл ~/**.bashrc** используется при запуске shell'a из уже работающей системы (читается при каждом новом вызове терминала), а ~/**.bash_profile** – при интерактивном входе (логине) в систему (в случае если вход осуществляется не через GUI либо в случае коннекта к системе через SSH), а также при запуске shell'a с опцией "**—login**" (например, "**bash —login**").

Краткое введение закончено, теперь пора узнать, как использовать эти переменные в шелле максимально эффективно.

Основные команды

Итак, еще раз, вот как выглядит командная строка и отвечающая за нее переменная PS1 на моей CentOS машине:

```
[gahcep@localhost ~]$ echo  $PS1
[\u@\h  \W]\$
```

В PS1 можно записывать не только предопределенные символьные последовательности, но и стандартные команды Linux, также объединяя их через **pipe** механизм. Все это чуть

позже, а пока вот какие последовательности можно использовать (весь список вы сможете найти в разделе “**Controlling the Prompt**”, я же приведу наиболее интересные и полезные, – в скобках даны примеры как выглядит строка при использовании последовательности):

Команды общего назначения

\h и **\H** — имя хоста, сокращенное и полное (**FQDN**, *Fully qualified domain name*), с учетом домена соответственно:

```
[ gahcep@localhost ~]$ echo  $PS1
[\u@\h  \W]\$
[gahcep@localhost ~]$ export PS1="[\u@\H \W]\$ "
[gahcep@localhost.localdomain ~]$
```

\u — имя пользователя **\n** — символ перевода строки **\$** — если эффективный идентификатор пользователя (*effective UID*) равен 0, печатает ‘#’, иначе ‘**$**’. Пояснение разницы между эффективным UID (eUID) и реальным (rUID) выходит за рамки этого поста, однако вот что следует знать: в обычных случаях eUID не используется, но, например, при попытке запущенного процесса открыть файл, система должна проверить его eUID для возможности или невозможности предоставления разрешения. Вот как выглядит строка при логине обычного пользователя:

```
[gahcep@localhost ~]$
```

А вот так при логине root'ом:

```
[root@localhost ~]#
```

[и **]** — начало и конец вывода непечатаемых символов. Эти символы обязательны при использовании цветовых кодов и при работе с курсором.
\e — ASCII escape символ, кодовое значение которого равно 033; \e идентичен 33. Символ используется в качестве начальной и конечной последовательности при задании цвета.

Дата и время

\d — простой, не настраиваемый формат “Weekday Month Day”:

```
gahcep@localhost ~]$ export PS1="[\d][\u@\h \W]\$ "
[Sat Feb 11][gahcep@localhost ~]$
```

\D{format} — настраиваемый формат времени и даты, вначале обрабатываемый функцией strftime, с последующей вставкой в shell:

```
[gahcep@localhost ~]$ export PS1="[\D{\%D \%H:\%m:\%S}][\u@\h \W]\$ "
[02/11/12 16:02:50][gahcep@localhost ~]$
```

\t — время в формате: 24 часа HH:MM:SS (17:34:23):

```
[gahcep@localhost ~]$ export PS1="[\t][\u@\h \W]\$ "
[17:34:23][gahcep@localhost ~]$
```

\T — время в формате: 12 часов HH:MM:SS (05:34:28):

```
[05:34:27][gahcep@localhost ~]$ export PS1="[\T][\u@\h \W]\$ "
[05:34:28][gahcep@localhost ~]$
```

\@ — время в формате: 12 часов am/pm (05:34 PM):

```
[gahcep@localhost ~]$ export PS1="[\@][\u@\h \W]\$ "
[05:34 PM][gahcep@localhost ~]$
```

\A — время в формате: 24 часа HH:MM (17:34):

```
[gahcep@localhost ~]$ export PS1="[\A][\u@\h \W]\$ "
[17:34][gahcep@localhost ~]$
```

Параметры терминала

\s — название вызванного шелла, точнее его `basename $0` часть (bash):

```
[gahcep@localhost ~]$ export PS1="[\u@\h \W] \s \$ "
[gahcep@localhost ~] bash $
```

\v — версия шелла, version (4.1):

```
[gahcep@localhost ~]$ export PS1="[\u@\h \W] \s-\v \$ "
[gahcep@localhost ~] bash-4.1 $
```

\V — полная релизная версия шелла, version + patchlevel (4.1.2):

```
[gahcep@localhost ~]$ export PS1="[\u@\h \W] \s-\V \$ "
[gahcep@localhost ~] bash-4.1.2 $
```

\l — печатает базовое имя терминального устройства (а по сути его номер), которое в данный момент привязано к шеллу. Что это значит? Рассмотрим на двух примерах:

#1 Терминал запущен в CentOS обычным путем. Запущена команда ps и изменена PS1:

```
[gahcep@localhost ~] $ ps
PID TTY TIME CMD
3321 pts/0 00:00:00 bash
4231 pts/0 00:00:00 ps
[gahcep@localhost ~] $ export PS1="[\u@\h \W] term:\l \$ "
[gahcep@localhost ~] term:0 $
```

Вывод списка процессов показывает, что имя терминального устройства привязанного к шеллу (т.е. к bash) равно 0: pts/**0**.

#2 Поднят коннект к CentOS'у по SSH. И вновь ps, затем изменение PS1:

```
[gahcep@localhost ~] $ ps
PID TTY TIME CMD
4087 pts/3 00:00:00 bash
4273 pts/3 00:00:00 ps
[gahcep@localhost ~] $ export PS1="[\u@\h \W] term:\l \$ "
[gahcep@localhost ~] term:3 $
```

Ну а в этом случае 0 сменился на 3.

Рабочая директория

\W — указывается только текущая директория – имя папки (RE). Домашняя директория обрамлена тильдой ‘~’:

```
[gahcep@localhost ~]$ export PS1="[\u@\h \W]\$ "
[gahcep@localhost ~]$ cd sources/RE/
[gahcep@localhost RE]$
```

\w — указывается полный путь без последнего слеша у директории (~/sources/RE, /usr/share/X11). Домашняя директория обрамлена тильдой ‘~’:

```
[gahcep@localhost ~]$ export PS1="[\u@\h \w]\$ "
[gahcep@localhost ~]$ cd sources/RE/
[gahcep@localhost ~/sources/RE]$ cd /usr/share/X11/
[gahcep@localhost /usr/share/X11]$
```

Нумерация команд

! — выводит под каким номером данная команда будет занесена в истории bash шелла (771,772):

```
[gahcep@localhost ~] $ export PS1="[\u@\h \W] cmd::\! \$ "
[gahcep@localhost ~] cmd::771 $ ls > /dev/null
```

```
[gahcep@localhost ~] cmd::772 $
```
— выводит под каким номером данная команда в текущем сеансе без привязки к истории. При запуске нового окна терминала, нумерация начнется с единицы:
```
[gahcep@localhost ~]$ export PS1="[\u@\h \W] cmd::\# \$ "
[gahcep@localhost ~] cmd::2 $ ls > /dev/null
[gahcep@localhost ~] cmd::3 $
```

Вот как этот номер коррелирует с номером команды в истории – сначала номер из истории, потом текущий (644 и 15, 645 и 16):

```
[gahcep@localhost ~] $ export PS1="[\u@\h \W] cmd h#\!/c#\# \$ "
[gahcep@localhost ~] cmd h#644/c#15 $ ls > /dev/null
[gahcep@localhost ~] cmd h#645/c#16 $
```

Краткий список команд, оформленный в виде таблицы:

Bash character	Bash character clarification
\a	an ASCII bell character (07)
\d	the date in "Weekday Month Date" format (e.g., "Tue May 26")
\e	an ASCII escape character (033)
\h	the FQDN hostname
\H	the hostname
\j	the number of jobs currently managed by the shell
\l	the basename of the shell's terminal device name
\n	newline
\r	carriage return
\s	the name of the shell, the basename of $0 (the portion followir final slash)
\t	the current time in 24-hour HH:MM:SS format
\T	the current time in 12-hour HH:MM:SS format
\@	the current time in 12-hour am/pm format
\u	the username of the current user
\v	the version of bash (e.g., 2.00)
\V	the release of bash, version + patchlevel (e.g., 2.00.0)
\w	the current working directory
\W	the basename of the current working directory
\!	the history number of this command
\#	the command number of this command
\$	if the effective UID is 0, a #, otherwise a $
\nnn	the character corresponding to the octal number nnn
\\	a backslash
\[	begin a sequence of non-printing characters, which could be u embed a terminal control sequence into the prompt
\]	end a sequence of non-printing characters

Помимо перечисленных выше символов-команд, возможен вариант встраивания простых и не очень команд в тело PS. Например, при переходе между директориями, хотелось бы сразу видеть количество файлов и директорий в них:

```
[gahcep@localhost ~]$ export PS1="[\u@\h] \W[\$(ls | wc -l)] \$ "
[gahcep@localhost] ~[14] $ cd /
[gahcep@localhost] /[20] $
```

Ответственные за это две команды: “**\$(ls | wc -l)**”. Как видите, результат одной команды подается на вход другой; **\$** в данном случае не является специальной символьной последовательностью. Также в PS1 можно вставлять пользовательские bash функции. Перепишем пример выше с использованием bash функции:

```
[gahcep@localhost ~] $ function count { ls | wc -l; }
[gahcep@localhost ~] $ export PS1="[\u@\h] \W[\$(count)] \$ "
[gahcep@localhost] ~[14] $ cd /
[gahcep@localhost] /[20] $
```

Если не устраивает написание функций прямо в командной строке, то функцию можно перенести в скрипт (который желательно лежал бы в одной из категорий $PATH) и спокойно использовать в PS1:

```
[gahcep@localhost ~]$ export PS1="[\u@\h] \W[\$(count.sh)] \$ "
[gahcep@localhost] ~[14] $ cd /
[gahcep@localhost] /[20] $
```

Содержимое **/usr/bin/count.sh** всего одна строчка: **ls | wc –l**.

Итак, мы рассмотрели самые значимые символьные команды, которые можно использовать для настройки строки ввода преимущественно в bash. О других шеллах чуть позже.

Раскрашиваем и перемещаем

Как было сказано ранее, набор команд (управляющих escape кодов) эмулятора терминала очень большой. Дальше мы посмотрим две группы (два подмножества): это работа с цветом и с курсором/экраном. Список всех возможных команд терминала VT100 можно найти в [5] и [6].

Раскрашиваем

Установка цвета осуществляется с помощью команды:

```
\e[X;Y;Zm
```

где

\e – escape символ (также равен 33, можно использовать вместо \e)
[– служебный символ
X – код атрибута текста(жирный, подчеркнутый, курсивом): **00**=none **01**=bold **02**=dim/half-bright **04**=underscore **05**=blink **07**=reverse **08**=concealed/hidden
Y – цветовой код текста (foreground). Здесь возможны варианты. Можно указать цвет **обычной***насыщенности* (*интенсивности*): **30**=black **31**=red **32**=green **33**=yellow **34**=blue **35**=magenta **36**=cyan **37**=white **39**=default или тот же цвет, но

с **повышенной** *насыщенностью* (*интенсивностью*): **90**=black **91**=red **92**=green **93**=yellow **94**=blue **95**=magenta **96**=cyan **97**=white
Z – цветовой код фона (background). Здесь также присутствует настройка *насыщенности*(*интенсивности*). Фон с цветом **обычной** насыщенности: **40**=black **41**=red **42**=green **43**=yellow **44**=blue **45**=magenta **46**=cyan **47**=white **49**=default или с **повышенной**: **100**=black **101**=red **102**=green **103**=yellow **104**=blue **105**=magenta **106**=cyan **107**=white
m – режим работы с цветом (чуть дальше мы рассмотрим команду работы с курсором, там вместо символа m символ f)

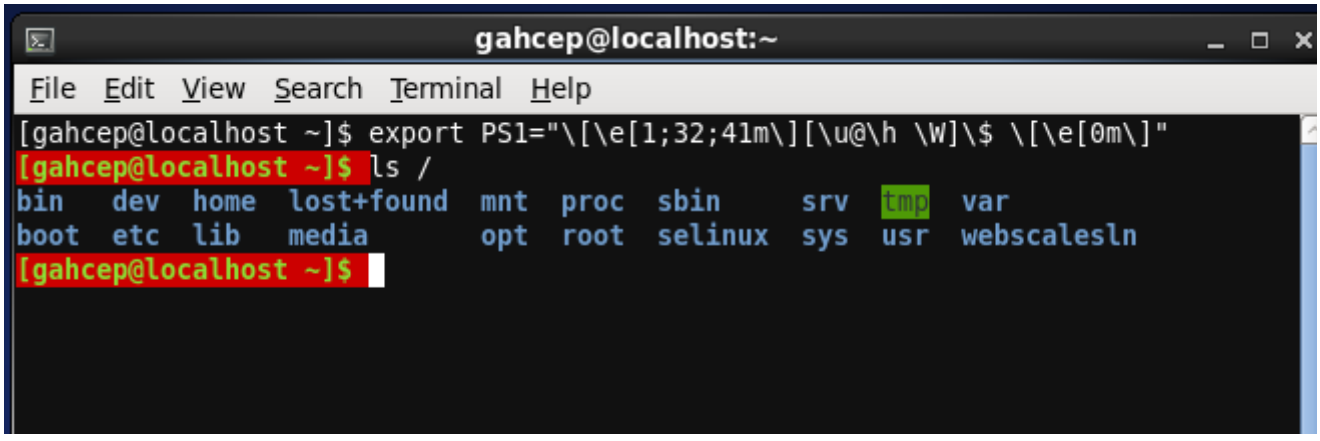
Сброс (если вы не хотите изменить весь вывод шелла – и команды и результат их работы) с помощью команды:

```
\e[0m
```

Заметьте, что диапазон цифр, шифрующий либо цвет текста, либо его свойство разный. Это означает, что можно задать как одно значение на позиции **X;Y;Z**, так и все три значения сразу. Обработка в любом случае будет корректная. Если указана одна цифра и она меньше 30, то значение будет интерпретироваться как свойства текста (bold, blink, пр.), если же величина больше 40, то очевидно, что задан цвет подложки, если же в диапазоне от 30 до 40, это – цвет текста. Если задано недопустимое число для дешифровки (например, 200 или 20), то если для него нет действительных значений, оно просто игнорируется.

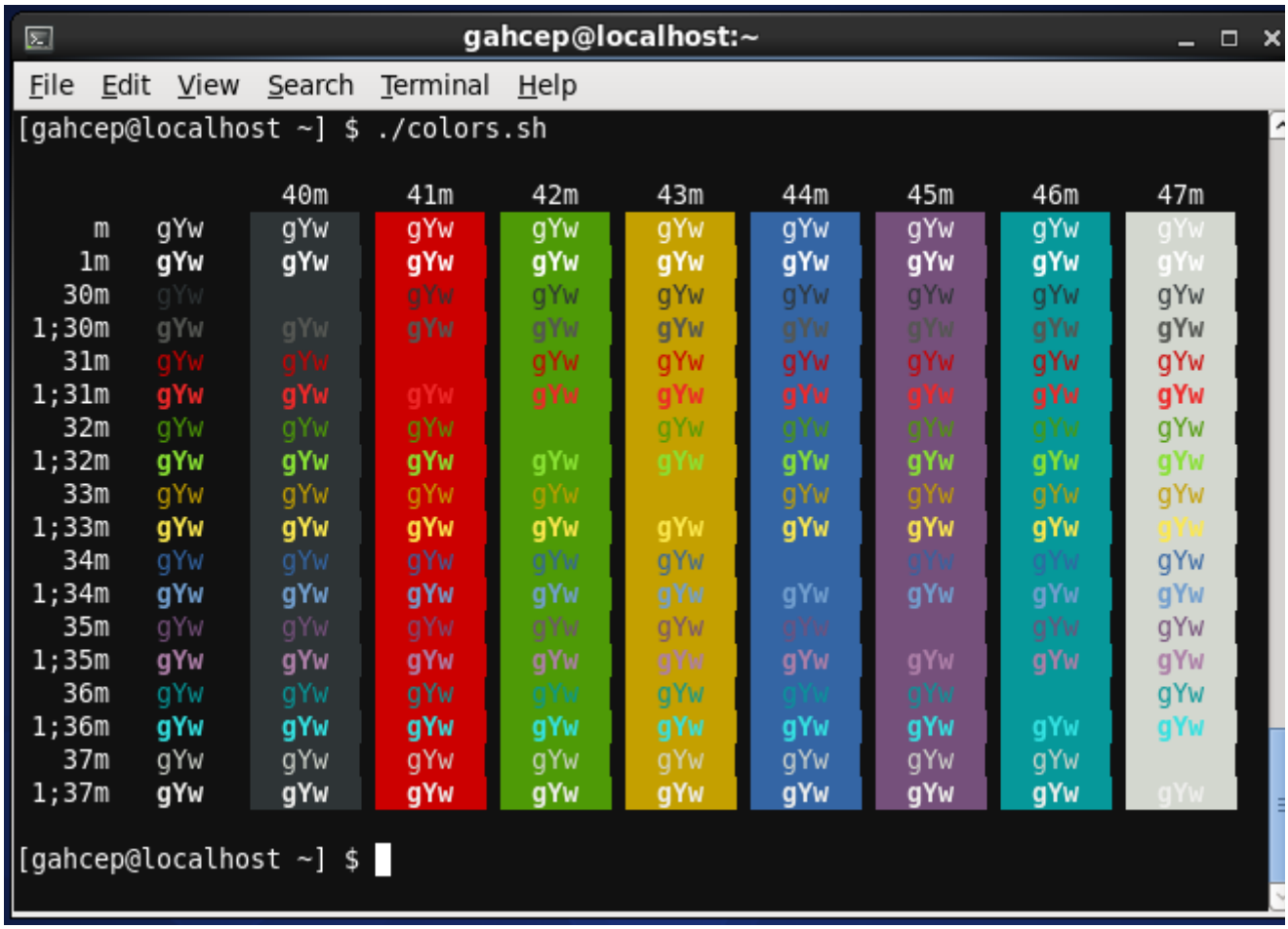
Таким образом, наличие отдельных непересекающихся диапазонов для трех категорий однозначно идентифицирует указанное значение.

Перед тем как продолжить, должен обратить внимание на обязательность экранирования **escape кодировки в случае использования в prompt строках** – для того, чтобы терминал корректно обработал последовательность: корректно посчитал длину строки и осуществил перевод на новую. При использовании кодов вкупе с командой “**echo –e**” его (экранирование) можно опустить. Экранирование заключается в обрамлении кода, задающего цвет в символы ‘**[**’ и ‘**]**’: “**\e[1;34m**” таким образом, превратится в “**[\e[1;34m]**”, а “**\e[s**” соответственно в “**[\e[s]**“. Далее по тексту в качестве примеров я буду опускать это обрамление, но в конкретных примерах кода оно будет присутствовать.
В качестве тренировки установим жирный (bold=01) шрифт зеленого (foreground green=32) цвета с красным (background red=41) фоном. Кодовая последовательность при этом будет “**\e[1;32;41m**”:



Если хотите использовать разные цвета для разных участков PS1, вперед, это не запрещено. Но согласитесь, что каждый раз вводить такую громоздкую последовательность довольно утомительно, кроме того, очень легко запутаться. Это вопрос можно легко решить, если вынести определения цветов в отдельный файл и подключить его скажем в **.bashrc**.

Ниже представлена цветовая таблица возможных сочетаний цвета шрифта и фона полученная с помощью скрипта на [4]. Замечу, что в ней отсутствует цветовая интенсивность (диапазон с 90 по 97 включительно для цвета шрифта и 100-107 для фона текста).



Перемещаем

Команда работы с курсором мало отличается от уже рассмотренной команды работы с цветами. Вот она:

- **\e** – escape символ (равен 33, можно использовать вместо \e)
- **[** – служебный символ
- **L** – позиция строки (Line)
- **C** – позиция колонки (Column)
- **X** – режим работы с курсором:
- **H, f** – перемещает курсор в заданную позицию на строку **L** в колонку **C**
- **A** – перемещает курсор вверх настрок. Используется только одно значение: \e[**L**A
- **B** – перемещает курсор вниз настрок. Используется только одно значение: \e[**L**B
- **C** – перемещает курсор вперед наколонок. Используется только одно значение: \e[**C**C
- **D** – перемещает курсор назад наколонок. Используется только одно значение: \e[**C**D

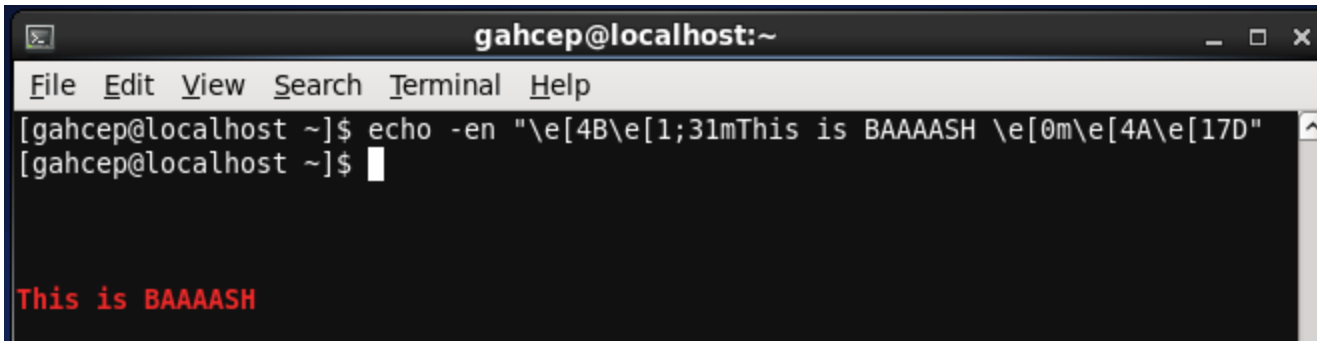
Помимо этих команд для работы с экраном и курсором есть большое количество других [5], например:

- **\e[K** – стереть линию до конца
- **\e[1K** – стереть линию с начала до текущей позиции курсора
- **\e[2K** – Стереть всю линию (позиция курсора останется неизменной)
- **\e[J** – Очистить экран от курсора вниз
- **\e[1J** – Очистить экран от курсора вверх
- **\e[2J** – Очистить весь экран
- **\e[s** – Сохранить позицию курсора
- **\e[u** – Восстановить позицию курсора

Имейте в виду, что не все вышеперечисленные команды работают во всех терминалах. Для демонстрации работы этих команд, предлагаю ввести не в переменную PS1, а в виде обычной команды следующую последовательность (идея заимствована из официального руководства **Bash-Prompt-HOWTO** на сайте [4]):

```
echo -en "\e[4B\e[1;31mThis is BAAAASH \e[0m\e[4A\e[17D"
```

И результат:



Что делает команда? Спускаемся на 4 строки вниз (**\e[4B**), печатаем красным цветом и жирным шрифтом фразу “**This is BAAAASH**” (\e[1;31mThis is BAAAASH \e[0m) после чего возвращаем позицию курсора обратно: 4 строки вверх (**\e[4A**) и на длину фразы влево (**\e[17D**).

Команда tput

В мире существует большое количество языков для работы с терминалами, у каждого набор команд, символьных последовательностей может быть просто огромен. Для общения с ними обычно существует некий промежуточный уровень предоставляющий определенный API. Этот API решает ряд низкоуровневых задач, как то: определение типа терминала, распознавание вводимых команд, поиск соответствия между введенными командами более высокого уровня и наборов кодов, которые поймет конкретный терминал. Одна из этих команд – **tput**. Изменение цвета через коды довольно неблагодарное занятие. Давайте попробуем переложить эту задачу на плечи **tput**. **tput** — команда, использующая базу данных terminfo для предоставления шеллу терминало-зависимых возможностей. При использовании команды можно указать тип терминала, но обычно это не делается, так как по-умолчанию тип берется из **$TERM** переменной. Далее мы посмотрим на часто используемые команды.

Имейте в виду, что возможности и функционал команд может отличаться от того же Solaris'a с его ksh: что-то может работать в шелле А, но не работать или работать отлично от ожиданий на терминале Б. Поэтому приведенный ниже обзор не призван перечислить

все команды tput, а лишь послужить введением в то, что есть tput и что можно делать с его помощью.

Рассмотрим основные команды tput.

Работа с цветом
0=black **1**=red **2**=green **3**=yellow **4**=blue **5**=magenta **6**=cyan **7**=white **9**=default

- tput **setab** [0–7,9] — установить цвет фона используя ANSI escape
- tput **setb** [0–7,9] — установить цвет фона
- tput **setaf** [0–7,9] — установить цвет текста используя ANSI escape
- tput **setf** [0–7,9] — установить цвет текста

Работа с текстом
tput [**bold**|**dim**|**smul**|**rmul**|**rev**|**sgr0**|**invis**]:

- **bold** – жирный шрифт
- **dim** – режим пониженной яркости
- **smul** – текст подчеркнут
- **rmul** – завершение режима smul
- **rev** – инвертирует цвет шрифта
- **sgr0** – сброс расширенных атрибутов текста
- **invis** – текст невидимый

Работа с курсором

- tput **cup R C** – устанавливает курсор (**CU**rsor **P**osition) в позицию “Row;Column”
- tput **cuf N** – двигает курсор (**CU**rsor **F**orward) на **N** позиций вправо
- tput **cub N** – двигает курсор (**CU**rsor **B**ack) на **N** позиций влево
- tput **cuu N** – двигает курсор (**CU**rsor **U**p) на **N** позиций вверх
- tput **cud N** – двигает курсор (**CU**rsor **D**own) на **N** позиций вниз
- tput **sc** – сохраняет позицию курсора
- tput **rc** – восстанавливает позицию курсора

Получение информации о терминале

- tput **lines** – количество строк (линий) в терминале
- tput **cols** – количество колонок в терминале

Очистка экрана

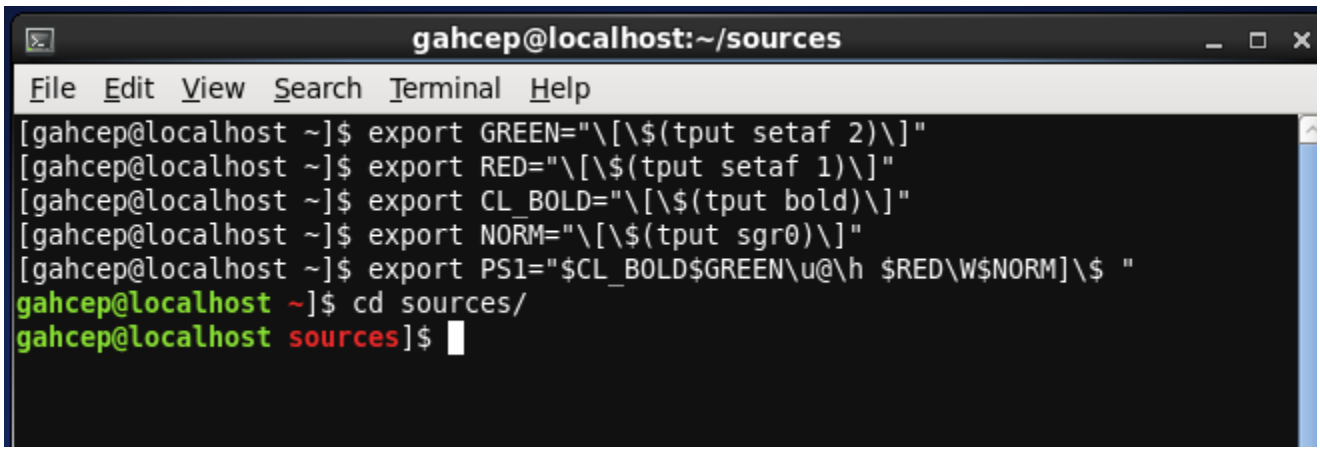
- tput [**reset**|**clear**] – очистка экрана, действует похоже команде clear

Имейте в виду, экранирование tput команд происходит по тем же правилам что и обычных escape команд шелла: “**\$(tput setaf 2)**” при использовании в PS1 превратится в “**[\$(tput setaf 2)]**”.

Рассмотрим пример использования команды. Выделим жирным зеленым цветом имя пользователя и хост, а жирным красным – текущую директорию:

```
[gahcep@localhost ~]$ export GREEN="\[\$(tput setaf 2)\]"
[gahcep@localhost ~]$ export RED="\[\$(tput setaf 1)\]"
[gahcep@localhost ~]$ export CL_BOLD="\[\$(tput bold)\]"
[gahcep@localhost ~]$ export NORM="\[\$(tput sgr0)\]"
[gahcep@localhost ~]$ export PS1="$CL_BOLD$GREEN\u@\h $RED\W$NORM]\$ "
[gahcep@localhost ~]$ cd sources/
[gahcep@localhost sources]$
```

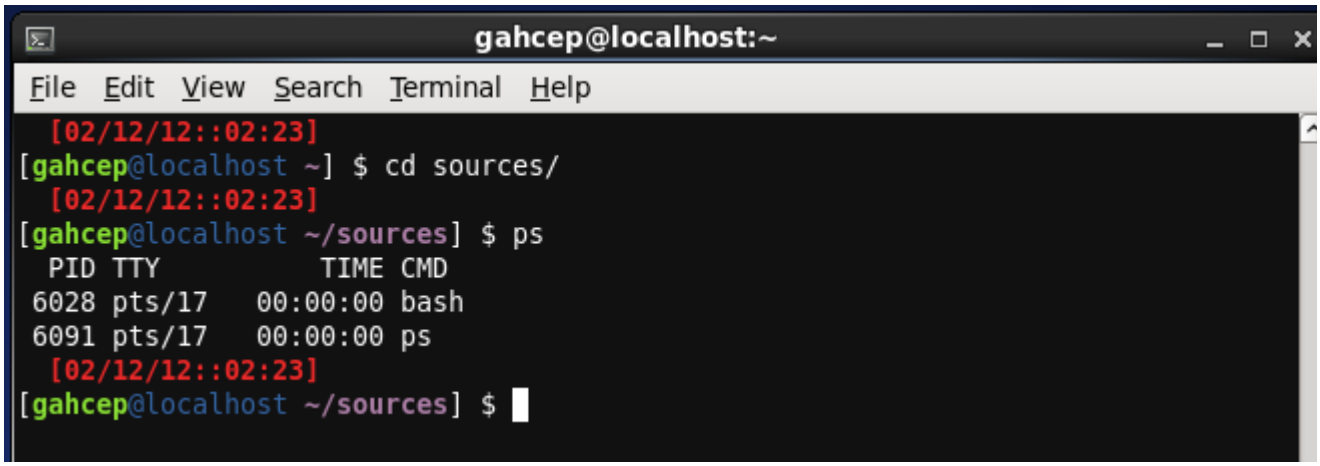
Вывод:



Тюнингуем шелл

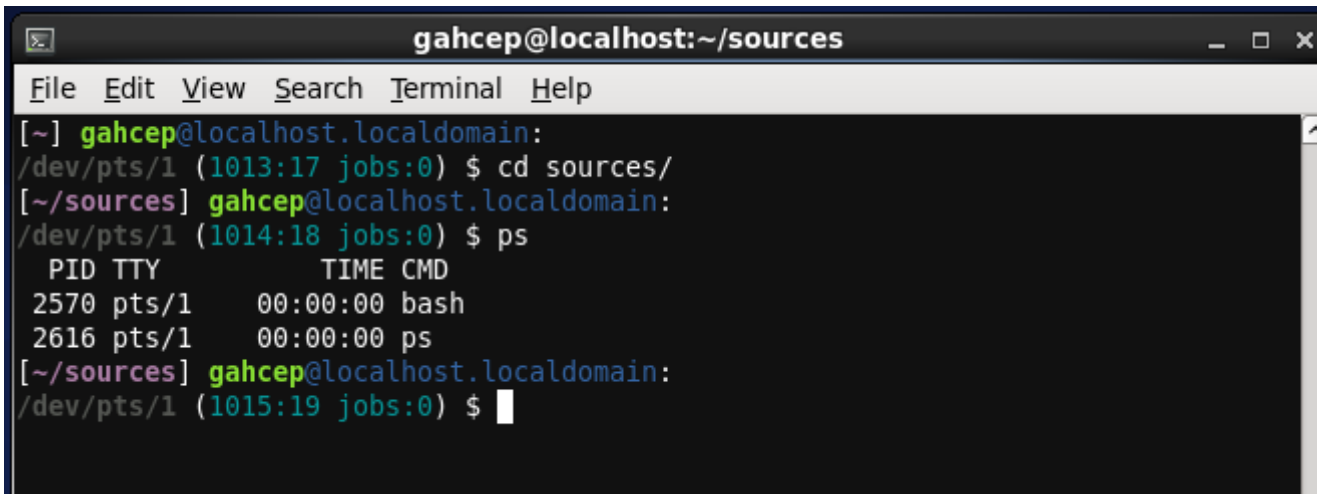
Прежде чем перейти к примерам, необходимо заметить, что для повышения удобочитаемости кода, все цветовые коды были сохранены в файл **.col_def** и подключены в **.bashrc**. Коды не экранированы, так что не забывайте добавлять ‘[’ и ‘]’: “**[$ANY_COLOR]**”. За основу взят файл с [7]. Ссылка на файл в конце статьи. Файл также доступен на pastebin.com.

#1 “Классика со временем”



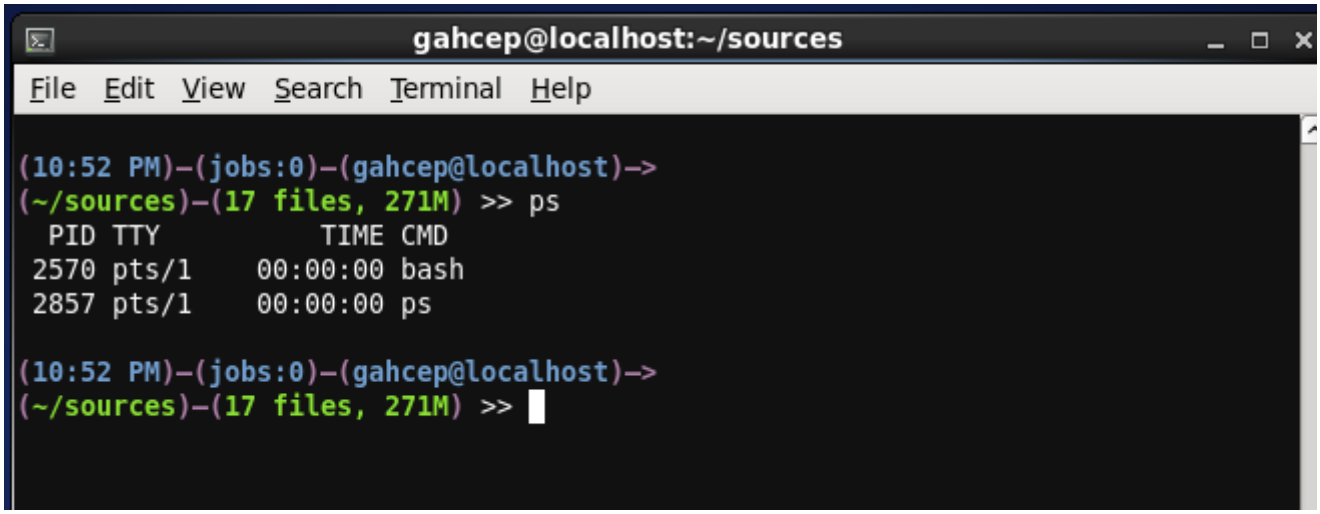

```
export PS1="[\[$BGreen\]\u\[$Blue\]@\h \[$BIPurple\]\w\[$ColorReset\]] \$ "
export PROMPT_COMMAND="echo -e '$(tput cuf 2)$BRed[$(date +%D::%m:%S)]$ColorReset'"
```

#2 “Больше информации”



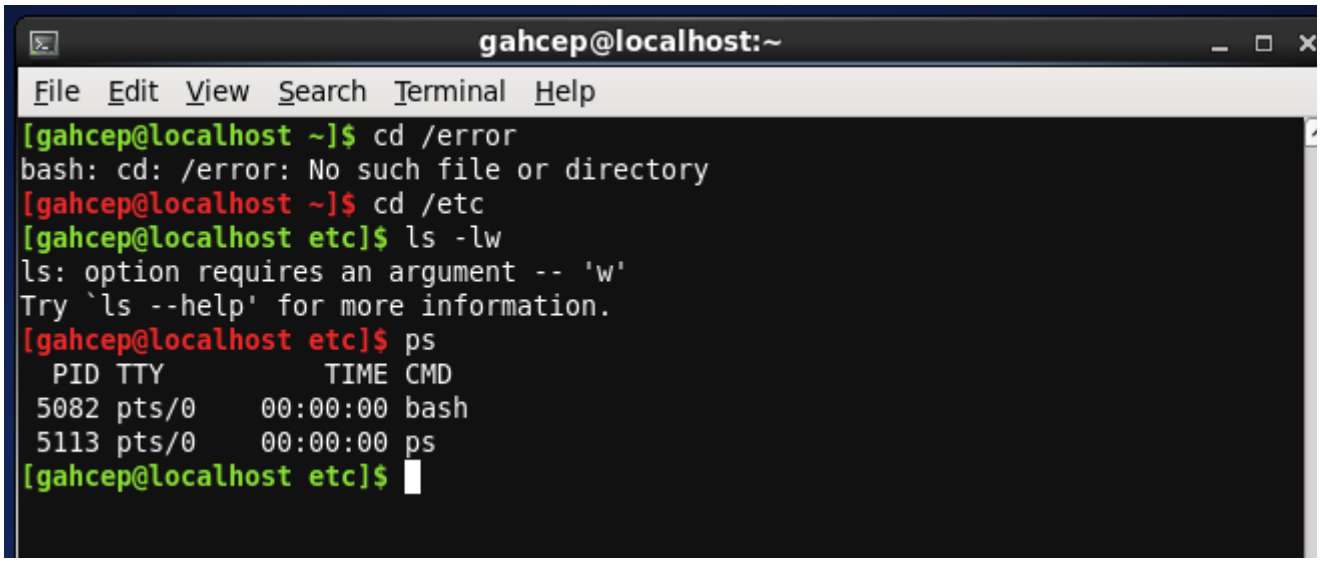
```
export PS1="[\[$BIPurple\]\w\[$ColorReset\]] \[$BGreen\]\u\[$Blue\]@\H
\[$ColorReset\]:\n\[$BBlack\]$(tty 2>/dev/null)\[$ColorReset\]
(\[$Cyan\]\!:\# jobs:\j\[$ColorReset\]) \$ "
```

#3 “Директории и файлы”



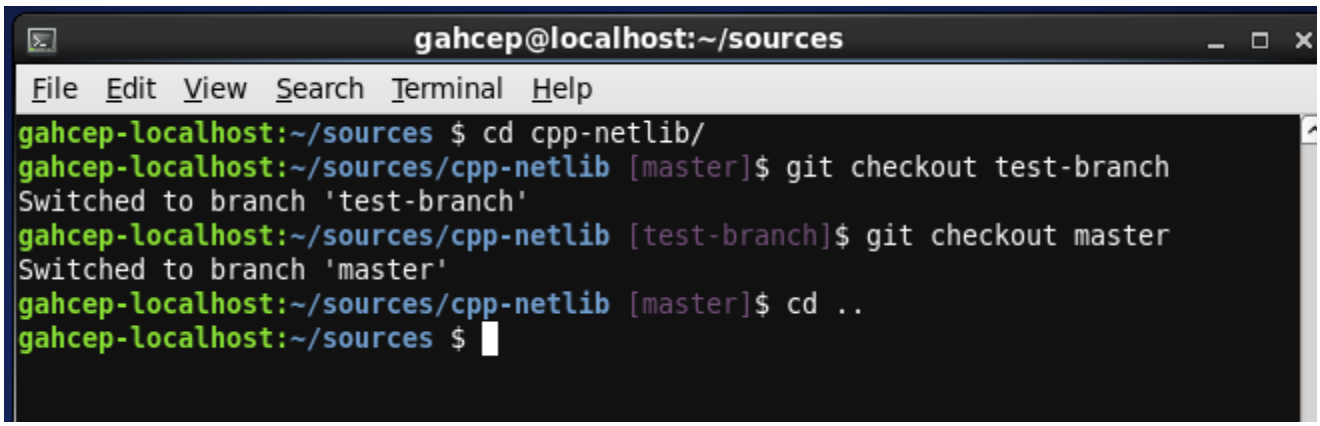

```
export PS1="\n\[$BIPurple\](\[$BIBlue\]\@\[$BIPurple\])—(\[$BIBlue\]jobs:
\j\[$BIPurple\])—(\[$BIBlue\]\u@\h\[$BIPurple\])—>\n(\[$BIGreen\]\w
\[$BIPurple\])—(\[$BIGreen\]$(ls -1 | wc -l) files, $(ls -lah |
grep -m 1 total | sed 's/total //')\[$BIPurple\])\[$ColorReset\] >> "
```

#4 “True or False. That’s the question”




```
export PS1=export PS1="[\u@\h \W]\$ \[$ColorReset\]"
export PROMPT_COMMAND='if [[ "$?" -eq "0" ]];
then echo -ne "$BGreen"; else echo -ne "$BRed"; fi;'
```

#5 “Ветви дерева Git”




```
export PS1="\[$BGreen\]\u-\h:\[$BBlue\]\w \[$Purple\]
\$(git branch --no-color 2> /dev/null |
sed -e '/^[^*]/d' -e 's/* \(.*\)/[\1]/')\[$ColorReset\]\$ "
```

Послесловие

Рассмотренные примеры, конечно же, далеко не все что можно сделать с интерфейсом командной строки. Читайте мануалы, смотрите форумы, экспериментируйте. Также советую почитать про возможности zsh шелла, у него есть много интересных особенностей, в том числе и в плане тюнинга командной строки. В конце вас ждут ссылки на использованные мною статьи и руководства, а также файлы, упомянутые в тексте. Спасибо, что нашли в себе силы и интерес прочитать эту статью. Если заметили ошибки, неточности, или просто хотите высказать критику, – вэлкам в комменты.

Приложение

Файлы

1. Файл с цветовыми определениями col_def.txt (при использовании уберите расширение txt).
2. Скрипт выводящий цветовую палитру терминала colors.txt (при использовании смените расширение на sh).

Ссылки

1. Терминал VT100 на Википедии
2. Определение xterm на Википедии
3. Раздел “Bash Variables” мануала по bash
4. Раздел “Colours” мануала “Bash Prompt HOWTO”
5. ANSI Escape sequences – VT100 / VT52
6. Terminal codes (ANSI/VT100) introduction
7. Color Bash Prompt на Вики ArchLinux
8. ANSI escape code на английской Википедии