How To Customize Bash Prompt in Linux

May 12, 2020

BASH LINUX

Home » SysAdmin » How To Customize Bash Prompt in Linux

Contents

Introduction

In Linux, much of your work occurs from a command prompt, also known as the **shell**, or **BASH** (Bourne-Again Shell). The shell interprets your commands and passes them to the operating system for execution.

This tutorial will show you how to **customize or change your Linux BASH prompt**.

Prerequisites

- A system running Linux
- Access to a command line/terminal
- A user account with **sudo** or **root** privileges

Default BASH Prompt

The default BASH prompt is the one you see when you first open a terminal or command line. It usually looks something like this:

```
username@hostname:~$
```

Alternatively, it may look like this:

```
(base) [username@localhost ~]$
```

The first part of the prompt tells you the user that's currently logged in. The second part identifies the hostname of the system.

The tilde sign ~ indicates that the current working directory is the current user's home directory.

The dollar sign **$** means the current user is a standard user.

A root user would be identified with a hash sign **#**.

Customize Bash Prompt In Linux

Like most Linux applications, BASH reads a configuration file to determine its behavior. This file is in the home directory:

```
~/.bashrc
```

Before you make any changes, create a backup copy of your configuration file. Open a terminal window, and enter the following:

```
cp ~/.bashrc ~/.bashrc.bak
```

Note: The system uses the **.bak** file extension to indicate a backup.

Change Bash Prompt in Linux Permanently

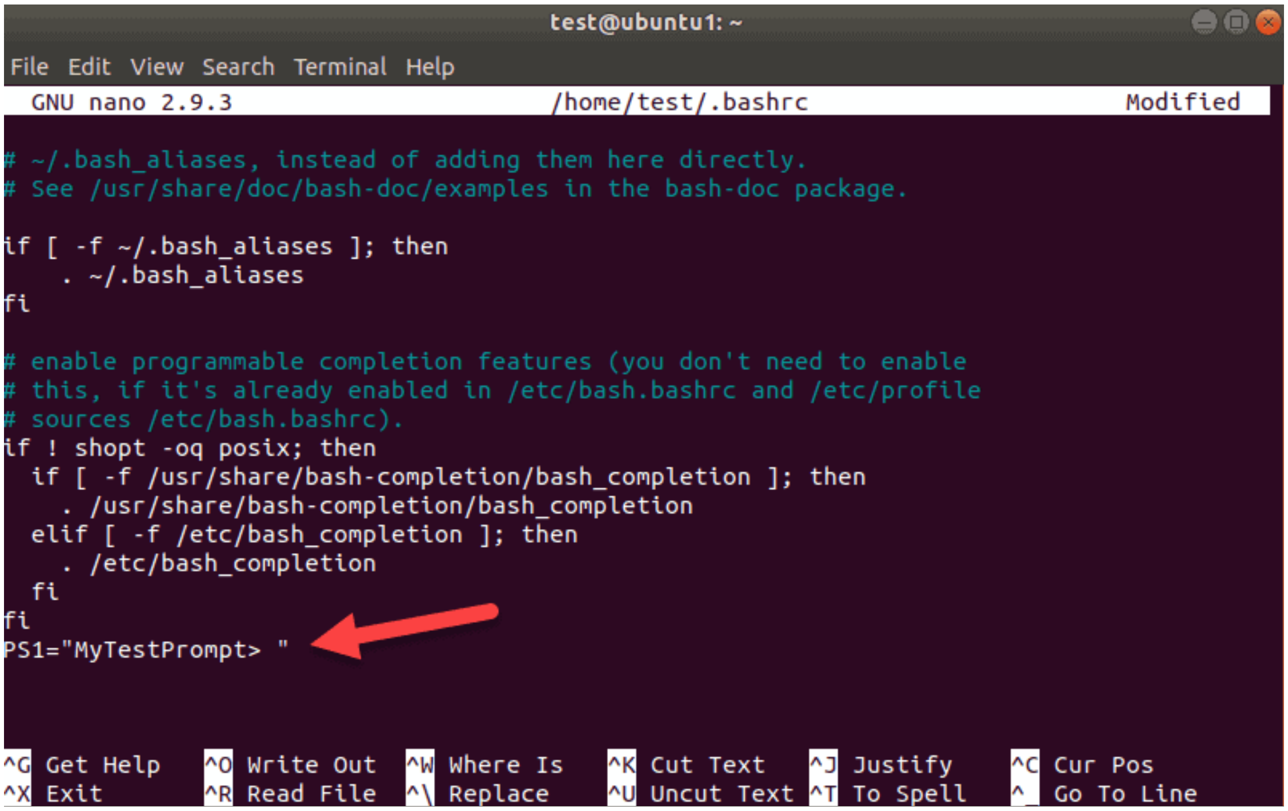
Open the BASH configuration file for editing:

```
sudo nano ~/.bashrc
```

In this file, you should see several different settings. Some of them are descriptive lines in blue, uncommented with a **#** sign. Some are white, which indicates that they are enabled.

Scroll to the bottom of the configuration file. Add the following line:

```
PS1="MyTestPrompt> "
```



```
  GNU nano 2.9.3                    /home/test/.bashrc                        Modified

# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
PS1="MyTestPrompt> "


^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

You can replace **`MyTestPrompt>`** with any string of text you like.

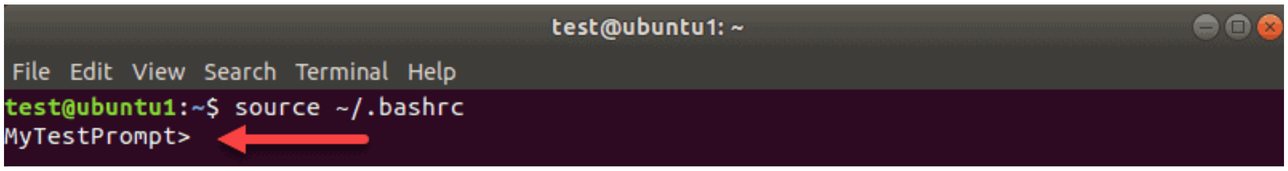
Save the file (**`ctrl-o`** > **Enter**) and exit (**`ctrl-x`**).

Refresh the BASH service to apply your changes. Enter the following:

```
source ~/.bashrc
```

Your command-line prompt should change to the following:

```
MyTestPrompt>
```



```
test@ubuntu1:~$ source ~/.bashrc
MyTestPrompt>
```

Note: Learn everything you need to know about working with Bash comments.

Create a Temporary Change to the BASH Prompt

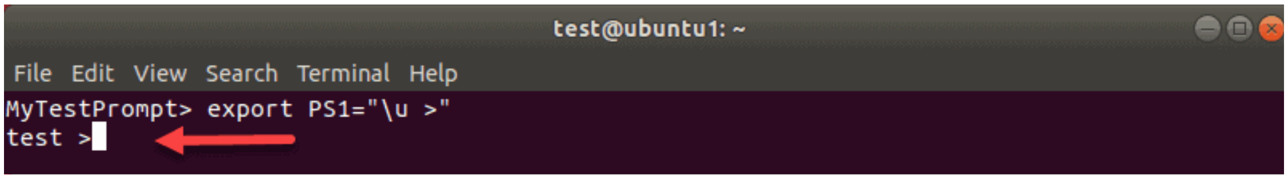
You can change the BASH prompt temporarily by using the **`export`** command. This command changes the prompt until the user logs out.

Set the BASH prompt to only display the username by entering the following:

```
export PS1="\u >"
```

The prompt should immediately change to look like this:

```
username >
```



```
MyTestPrompt> export PS1="\u >"
test >
```

You can reset the prompt by logging out, then logging back in.

Popular Custom Options for BASH Prompts

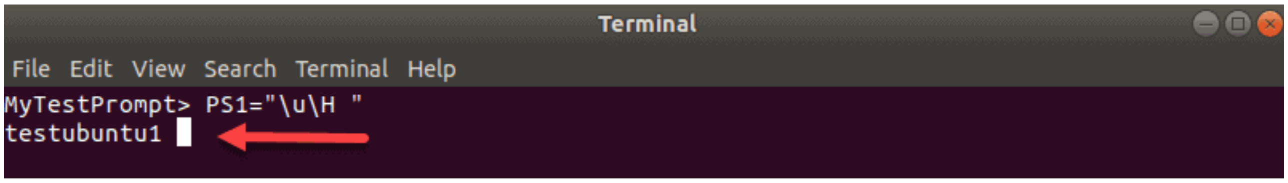
You can use these options in either method – temporarily with the **`export`** command, or permanently by editing the **`~/.bashrc`** file.

Display Username and Domain Name

Use the **–H** option to display a a full hostname:

```
export PS1="\u\H "
```

You should see the hostname in the prompt.



```
MyTestPrompt> PS1="\u\H "
testubuntu1
```

Add Special Characters

You can add special characters to the prompt by placing them in order around the special options:

```
export PS1="\u@\H :"
```

This should display the following:

```
username@domain:
```

Note: We recommend ending the prompt with a special character or space. You should also place a space, colon, or angle-bracket just before the final quote mark. This method helps users tell the difference between the prompt and the command they're typing.

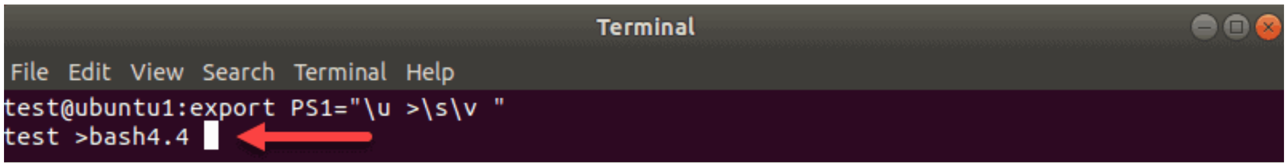
Display Username Plus Shell Name and Version

Enter the following to show username, shell name, and version:

```
export PS1="\u >\s\v "
```

The prompt should change to the following:

```
username >bash4.4
```



```
test@ubuntu1:export PS1="\u >\s\v "
test >bash4.4
```

Add Date and Time to The BASH Prompt

Use the following options to display different formats for date and time:

- **d** – Displays today's date in [weekday]/[month]/[day]

```
export PS1="\u@\H>\d "
```

- **t** – Displays the current time in 24-hour notation

```
export PS1="\u@\H>\t "
```

- **T** – Displays the current time in 12-hour notation

```
export PS1="\u@\H>\T "
```

- **A** – Displays the current time in 24-hour notation, with just hours and minutes

```
export PS1="\u@\H>\A "
```

Note: The **\u@\H** options preceding the date and time option add the username and full domain name.

Hide All Information in the BASH Prompt

Use this to prevent usernames or hostnames from displaying at the prompt:

```
export PS1="\W > "
```

You should see the following:

```
~ >
```

Differentiate Root User From Normal User

The normal BASH prompt displays a **$** sign for a normal user. If you log in as a root user, a **#** sign is displayed. Use the **$** code to indicate that the current user is not a root user:

```
export PS1="\u@\H \W:\$ "
```

More BASH Prompt Options

Here is a list of most of the options you can use for the BASH prompt.

Some of these commands may not work on all versions of Linux.

- **`\a`** – A bell character
- **`\d`** – Date (day/month/date)
- **`\D{format}`** – Use this to call the system to respond with the current time
- **`\e`** – Escape character
- **`\h`** – Hostname (short)
- **`\H`** – Full hostname (domain name)
- **`\j`** – Number of jobs being managed by the shell
- **`\l`** – The basename of the shells terminal device
- **`\n`** – New line
- **`\r`** – Carriage return
- **`\s`** – The name of the shell
- **`\t`** – Time (hour:minute:second)

- **`\@`** – Time, 12-hour AM/PM
- **`\A`** – Time, 24-hour, without seconds
- **`\u`** – Current username
- **`\v`** – BASH version
- **`\V`** – Extra information about the BASH version
- **`\w`** – Current working directory ($HOME is represented by ~)
- **`\W`** – The basename of the working directory ($HOME is represented by ~)
- **`\!`** – Lists this command's number in the history
- **`\#`** – This command's command number
- **`\$`** – Specifies whether the user is root (#) or otherwise ($)
- **`\\`**– Backslash
- **`\[`** – Start a sequence of non-displayed characters (useful if you want to add a command or instruction set to the prompt)
- **`\]`** – Close or end a sequence of non-displayed characters

How to Change BASH Prompt Color

You can change the text color of your BASH prompt. For example, to temporarily change the text of your BASH prompt to green, enter the following:

```
export PS1="\e[0;32m[\u@\h \W]\$ \e[0m"
```

Your prompt should have the same text as normal but be colored green.

Here's a breakdown of the commands:

• **`\e[`** – Begin color changes
• **`0;32m`** – Specify the color code
• **`[\u@\h \W]\$`** – This is the code for your normal BASH prompt (`username@hostname Workingdirectory $`)
• **`\e[0m`** – Exit color-change mode

The first number in the color code specifies the typeface:

- **0** – Normal
- **1** – Bold (bright)
- **2** – Dim
- **4** – Underlined

The second number indicates the color you want:

- **30** – Black
- **31** – Red
- **32** – Green
- **33** – Brown
- **34** – Blue
- **35** – Purple
- **36** – Cyan
- **37** – Light gray

Additionally, if you combine the bright option with a color code, you get a lighter version of that color. For example, if you use color code **`1;32`**, you would get light green instead of the normal green. If you use **`1;33`**, you get yellow instead of brown.

How to Reset BASH Changes to Default Settings

There are two ways to reset the changes. For temporary changes (using the **`export PS1 =""`** command), you can reset the default by logging out.

If you edited the **`\.bashrc`** file to make permanent changes, there are two methods to revert to default settings:

- Render your changes as comments by editing the file and adding a **#** before each change you made.
- Restore default settings from your backup by entering:

```
sudo cp ~/.bashrc.bak ~/.bashrc
```

Understanding Different Parts of BASH

Prompt

Before you continue, reset your BASH prompt to the default. If you used the **`export`** command, log out and log back in. If you edited your **`~/.bashrc`** file, place a **#** sign before each edit you made and save the file.

The BASH prompt contains four different values: **PS1, PS2, PS3, and PS4**.

The PS stands for **Prompt Statement**. So far, we've been working with the PS1 value. To see the current PS1 value, enter the following:

```
echo $PS1
```

Depending on the system, the terminal returns something like this for the default settings:

You might recognize the **`\u@\h`** options as the **username** and **host**. The **`w`** option displays the current working directory.

Now, display the PS2 value:

```
echo $PS2
```

The system should display just an angle-bracket:

```
>
```

Repeating the same command for PS3 should be blank.

For PS4, you'll see a **+** sign.

Here are the different meanings for the different parts of the BASH prompt:

- **PS1** – This is the primary prompt display. This is where you set special characters or important information.
- **PS2** – This is the secondary prompt string. This is usually set as a divider between the prompt display and the text entry. It is also used to display when a long command is broken into sections with the \ sign.
- **PS3** – This is the prompt for the **`select`** command.
- **PS4** – This is the prompt for running a shell script in debug mode.

Under most circumstances, you'll be working with just the **PS1** option and maybe the **PS2** option as well.

Note: Refer to our article Bash Function & How to Use It to learn how to optimize your coding and scripting.

Conclusion

You should now be able to customize your BASH prompt. You can combine the commands and options to get the look and feel you want.

Every time you start a shell session in Linux, the system goes through configuration files and sets up the environment accordingly. Check out our guide on how to set, view, and unset environment variables in Linux.

Next, Vim users may want to customize Vim and apply custom color schemes.

Goran Jevtic

Goran combines his leadership skills and passion for research, writing, and technology as a Technical Writing Team Lead at phoenixNAP. Working with multiple departments and on various projects, he has developed an extraordinary understanding of cloud and virtualization technology trends and best practices.

Was this article helpful? Yes No

Next you should read

SysAdmin, Web Servers

How to Use wget Command With

E
x
a
m
pl
e
s

SysAdmin, Web Servers

How to List Installed Packages on Ubuntu

June 12, 2019

Having a precise list of installed packages helps system admins maintain...

READ MORE

SysAdmin

How To Use grep Command In Linux/UNIX

March 28, 2019

This guide details the most useful grep commands for Linux...

READ MORE

SysAdmin

H
o
w
t
o
u
s
e
a
p
t
P
a
c
k
a
g
e
M
a
n
a
g
e
r
o
n
U
b
u

n t u Li n u x

Ja nu ary 7, 20 19

In Li n ux , sp ec ial to ol s w er e de ve lo pe d fo r m

a
n
a
gi
n
g
a
p
pli
ca
ti
o
ns
.
A
p
pli
ca
ti
o
n..
.
**RE
A
D
M
O
RE**

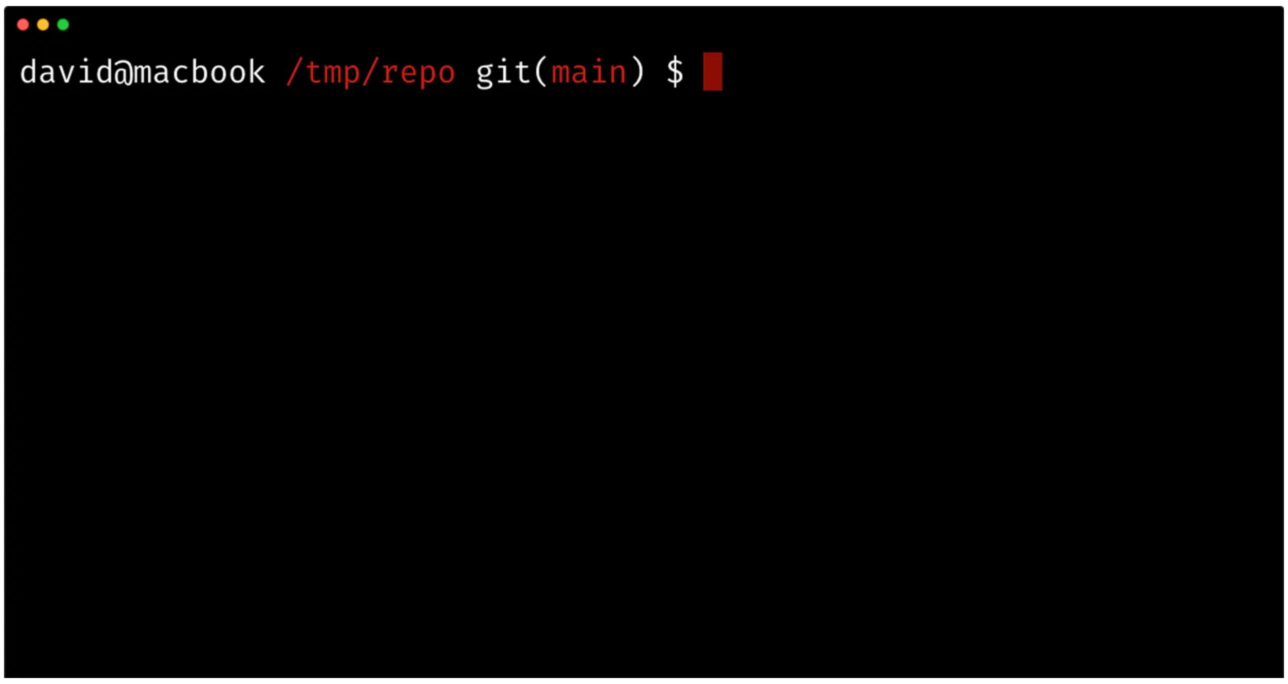
Live Chat   Get a Quote   Support | 1-855-330-1509   Sales | 1-877-588-5918

Privacy Policy   GDPR   Sitemap

Menu

Follow Me

@voraciousdev



```
david@macbook /tmp/repo git(main) $
```

A Guide to Customizing the Zsh Shell Prompt

David R. Myers

Mar 11, 2021 · 6 min read

The goal of this article is to teach you just enough about the shell prompt to make some helpful customizations.

What is the shell prompt?

The prompt is the bit of text that shows up in our shells to indicate that we can interact with them. The prompt usually gives us some details about the current shell session such as username, machine name, current directory, and some kind of prompt termination token. An example might look something like this.

COPY

```
david@macbook /tmp $
```

All of this information can be customized through the shell's prompt strings. Each shell has specific escape sequences that must be used. This just means that we have to use different tokens to represent things such as username, color formatting, etc. For the purposes of this article, we will be using Zsh, and we can confirm that is our current shell by running this.

COPY

```
echo $0 # /bin/zsh
```

Introducing PS1

The primary prompt string is stored in a variable called `PS1`. There are five prompt strings in total, so the trailing number denotes its responsibility. The primary prompt string is the one that is printed to `stdout` when the shell is waiting for a new command, and it is therefore the one we probably see most frequently. Unlike a typical variable, these prompt strings undergo expansion *every time the*

COPY

```
PS1='%n@%m %/ $ ' # david@macbook /tmp $
```

We will dive into this more below.

Breaking it down

The first thing to note is the use of single quotes `''`. We can use double quotes, but we have to remember to escape all of the expressions that we want to re-evaluate each time the prompt is displayed. Double quoted strings undergo normal expansion and substitution *before* being stored in a variable, so evaluation would only happen once per shell session rather than once per prompt. It doesn't make a difference for our simple example above, but it starts to matter when we introduce things like variables and shell expressions. More on that later.

Next, let's take a look at each of the escape sequences in our prompt string. The first one is `%n`, and it represents our username. The next escape sequence is `%/`, and it represents the current directory. For these examples, we will assume the username is `david` and the current directory is `/tmp`. For more on Zsh prompt expansion, check out the docs.

How do we customize it?

We are going to build our prompt from scratch, so let's start with some basic information. We talked about the `PS1` variable above, but we didn't talk about where it needs to be defined. For Zsh, we will use `~/.zshrc` to set the `PS1` variable. This file is loaded at the beginning of every shell session, so it will ensure our configuration is respected for all sessions and not just the current one. Every shell has a default prompt string, and we can view the raw source by echoing the `PS1` variable (e.g. `echo $PS1`). While we can build on top of the existing one, for the purposes of learning, we are going to build ours from

This prompt does its primary job of telling us the shell is ready for input, but we can make it much more useful. Let's start by adding our username `%n`, our machine name `%m`, and our current directory `%/`.

COPY

```
# ~/.zshrc
PS1='%n@%m %/ $ ' # david@macbook /tmp $
```

That's looking a lot more like what we might expect from a typical shell prompt.

Adding some color

Now that we have a basic prompt, we can spice it up with some color. To change the color of our prompt, we need to use a new type of sequence. Many modern terminals support 256 colors, and while some of these can be referenced by name, most will need to be referenced by their color code - ranging from 0 to 255. Colors are applied via start and stop sequences indicated by `%F{}` and `%f`, respectively. Let's put this into practice by coloring our current directory red.

COPY

```
# ~/.zshrc
PS1='%n@%m %F{red}%/%f $ ' # david@macbook /tmp $
```

Adding the current Git branch

Another useful piece of information which is not typically displayed by default is version control. Not all directories are version controlled, so this info will only show up when it is relevant. For this guide, we will use Git, but Zsh actually does support a few others. We are going to add

This block is what actually enables and autoloads `vcs_info` before each rendering of the shell prompt. It stores the data in a new variable called `vcs_info_msg_0_`. If we use this variable in our prompt string, it will display some basic info such as our version control system and the current branch (e.g. `(git)-[main]-`). To format this info, we can use `zstyle` like so.

COPY

```
# ~/.zshrc
zstyle ':vcs_info:*' formats ' %s(%b)' # git(main)
```

The `zstyle` format string has its own tokens that are expanded in the prompt string. The first token `%s` represents the current version control system. The next token `%b` represents the current branch name. For more information on available tokens, check out the docs. Finally, we can combine everything we've learned and add a bit of color to the branch name.

COPY

```
# ~/.zshrc
autoload -Uz vcs_info # enable vcs_info
precmd () { vcs_info } # always load before displaying the pr
zstyle ':vcs_info:*' formats ' %s(%F{red}%b%f)' # git(main)

PS1='%n@%m %F{red}%/%f$vcs_info_msg_0_ $ ' # david@macbook /t
```

Final result

status information, and how to apply color to these various pieces of information.

Closing

I hope this article helped shed some light on the shell prompt syntax. It was a lot of fun to write while experimenting with my own prompt (which you can see at the top of this page). As a final note, I would love to share octo - a hackable, offline-first markdown editor for notes, code snippets, and writing that runs entirely in-browser. It's free and open source, and working on it gives me a lot of ideas and inspiration for the articles I write. Thank you, and happy coding. ✌️

#bash #zsh #tutorial #web-development

WRITTEN BY
David R. Myers

Follow

Senior Software Engineer sharing resources, tips, and lessons learned | Building octo.app in public | Black Lives Matter

❤️ 2 👍 1 🦄 1 👏 1 🍺 1 🏆 1 😍 1 💰 1
🎉 1 🚀 1

MORE ARTICLES

David R. Myers

A Practical Cheat Sheet for CSS Flexbox (Containers)

I originally posted this Flexbox cheat sheet on Twitter, but the response was so positive that I dec...

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13 Prompt Expansion

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13.1 Expansion of Prompt Sequences

Prompt sequences undergo a special form of expansion. This type of expansion is also available using the `-P` option to the `print` builtin.

If the `PROMPT_SUBST` option is set, the prompt string is first subjected to *parameter expansion*, *command substitution* and *arithmetic expansion*. See Expansion.

Certain escape sequences may be recognised in the prompt string.

If the `PROMPT_BANG` option is set, a '`!`' in the prompt is replaced by the current history event number. A literal '`!`' may then be represented as '`!!`'.

If the `PROMPT_PERCENT` option is set, certain escape sequences that start with '`%`' are expanded. Many escapes are followed by a single character, although some of these take an optional integer argument that should appear between the '`%`' and the next character of the sequence. More complicated escape sequences are available to provide conditional expansion.

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13.2 Simple Prompt Escapes

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13.2.1 Special characters

`%%`

A '`%`'.

`%)`

A '`)`'.

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13.2.2 Login information

`%l`

The line (tty) the user is logged in on, without '`/dev/`' prefix. If the name starts with '`/dev/tty`', that prefix is stripped.

`%M`

The full machine hostname.

`%m`

The hostname up to the first '`.`'. An integer may follow the '`%`' to specify how many components of the hostname are desired. With a negative integer, trailing components of the hostname are shown.

`%n`

`$USERNAME`.

`%y`

The line (tty) the user is logged in on, without '`/dev/`' prefix. This does not treat '`/dev/tty`' names specially.

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13.2.3 Shell state

`%#`

A '`#`' if the shell is running with privileges, a '`%`' if not. Equivalent to '`%(!.#.%%)`'. The definition of 'privileged', for these purposes, is that either the effective user ID is zero, or, if POSIX.1e capabilities are supported, that at least one capability is raised in either the Effective or Inheritable capability vectors.

`%?`

The return status of the last command executed just before the prompt.

`%_`

The status of the parser, i.e. the shell constructs (like '`if`' and '`for`') that have been started on the command line. If given an integer number that many strings will be printed; zero or negative or no integer means print as many as there are. This is most useful in prompts `PS2` for continuation lines and `PS4` for debugging with the `XTRACE` option; in the latter case it will also work non-interactively.

`%^`

The status of the parser in reverse. This is the same as '`%_`' other than the order of strings. It is often used in `RPS2`.

`%d`
`%/`

Current working directory. If an integer follows the '`%`', it specifies a number of trailing components of the current working directory to show; zero means the whole path. A negative integer specifies leading components, i.e. `%-1d` specifies the first component.

`%~`

As `%d` and `%/`, but if the current working directory starts with `$HOME`, that part is replaced by a '`~`'. Furthermore, if it has a named directory as its prefix, that part is replaced by a '`~`' followed by the name of the directory, but only if the result is shorter than the full path; Filename Expansion.

`%e`

Evaluation depth of the current sourced file, shell function, or `eval`. This is incremented or decremented every time the value of `%N` is set or reverted to a previous value, respectively. This is most useful for debugging as part of `$PS4`.

`%h`
`%!`

Current history event number.

`%i`

The line number currently being executed in the script, sourced file, or shell function given by `%N`. This is most useful for debugging as part of `$PS4`.

`%I`

The line number currently being executed in the file `%x`. This is similar to `%i`, but the line number is always a line number in the file where the code was defined, even if the code is a shell function.

`%j`

The number of jobs.

`%L`

The current value of `$SHLVL`.

`%N`

The name of the script, sourced file, or shell function that zsh is currently executing, whichever was started most recently. If there is none, this is equivalent to the parameter `$0`. An integer may follow the '`%`' to specify a number of trailing path components to show; zero means the full path. A negative integer specifies leading components.

`%x`

The name of the file containing the source code currently being executed. This behaves as `%N` except that function and eval command names are not shown, instead the file where they were defined.

`%c`
`%.`
`%C`

Trailing component of the current working directory. An integer may follow the '`%`' to get more than one component. Unless '`%C`' is used, tilde contraction is performed first. These are deprecated as `%c` and `%C` are equivalent to `%1~` and `%1/`, respectively, while explicit positive integers have the same effect as for the latter two sequences.

13.2.4 Date and time

`%D`

The date in *yy*-*mm*-*dd* format.

`%T`

Current time of day, in 24-hour format.

`%t`
`%@`

Current time of day, in 12-hour, am/pm format.

`%*`

Current time of day in 24-hour format, with seconds.

`%w`

The date in *day*-*dd* format.

`%W`

The date in *mm*/*dd*/*yy* format.

`%D{`*string*`}`

string is formatted using the `strftime` function. See man page strftime(3) for more details. Various zsh extensions provide numbers with no leading zero or space if the number is a single digit:

`%f`

a day of the month

`%K`

the hour of the day on the 24-hour clock

`%L`

the hour of the day on the 12-hour clock

In addition, if the system supports the POSIX `gettimeofday` system call, `%.` provides decimal fractions of a second since the epoch with leading zeroes. By default three decimal places are provided, but a number of digits up to 9 may be given following the `%`; hence `%6.` outputs microseconds, and `%9.` outputs nanoseconds. (The latter requires a nanosecond-precision `clock_gettime`; systems lacking this will return a value multiplied by the appropriate power of 10.) A typical example of this is the format '`%D{%H:%M:%S.%.}`'.

The GNU extension `%N` is handled as a synonym for `%9.`.

Additionally, the GNU extension that a '`-`' between the `%` and the format character causes a leading zero or space to be stripped is handled directly by the shell for the format characters `d`, `f`, `H`, `k`, `l`, `m`, `M`, `S` and `y`; any other format characters are provided to the system's strftime(3) with any leading '`-`' present, so the handling is system dependent. Further GNU (or other) extensions are also passed to strftime(3) and may work if the system supports them.

13.2.5 Visual effects

`%B` (`%b`)

Start (stop) boldface mode.

`%E`

Clear to end of line.

`%U` (`%u`)

Start (stop) underline mode.

`%S` (`%s`)

Start (stop) standout mode.

`%F` (`%f`)

Start (stop) using a different foreground colour, if supported by the terminal. The colour may be specified two ways: either as a numeric argument, as normal, or by a sequence in braces following the `%F`, for example `%F{red}`. In the latter case the values allowed are as described for the `fg` `zle_highlight` attribute; Character Highlighting. This means that numeric colours are allowed in the second format also.

`%K` (`%k`)

Start (stop) using a different bacKground colour. The syntax is identical to that for `%F` and `%f`.

`%{...%}`

Include a string as a literal escape sequence. The string within the braces should not change the cursor position. Brace pairs can nest.

A positive numeric argument between the `%` and the `{` is treated as described for `%G` below.

`%G`

Within a `%{...%}` sequence, include a 'glitch': that is, assume that a single character width will be output. This is useful when outputting characters that otherwise cannot be correctly handled by the shell, such as the alternate character set on some terminals. The characters in question can be included within a `%{...%}` sequence together with the appropriate number of `%G` sequences to indicate the correct width. An integer between the '`%`' and '`G`' indicates a character width other than one. Hence `%{`*seq*`%2G%}` outputs *seq* and assumes it takes up the width of two standard characters.

Multiple uses of `%G` accumulate in the obvious fashion; the position of the `%G` is unimportant. Negative integers are not handled.

Note that when prompt truncation is in use it is advisable to divide up output into single characters within each `%{...%}` group so that the correct truncation point can be found.

[<<] [<] [Up] [>] [>>]   [Top] [Contents] [Index] [?]

13.3 Conditional Substrings in Prompts

`%v`

The value of the first element of the `psvar` array parameter. Following the '`%`' with an integer gives that element of the array. Negative integers count from the end of the array.

`%(`*x*`.`*true-text*`.`*false-text*`)`

Specifies a ternary expression. The character following the *x* is arbitrary; the same character is used to separate the text for the 'true' result from that for the 'false' result. This separator may not appear in the *true-text*, except as part of a %-escape sequence. A '`)`' may appear in the *false-text* as '`%)`'. *true-text* and *false-text* may both contain arbitrarily-nested escape sequences, including further ternary expressions.

The left parenthesis may be preceded or followed by a positive integer *n*, which defaults to zero. A negative integer will be multiplied by -1, except as noted below for '`l`'. The test character *x* may be any of the following:

`!`

True if the shell is running with privileges.

`#`

True if the effective uid of the current process is *n*.

`?`

True if the exit status of the last command was *n*.

`_`

True if at least *n* shell constructs were started.

`C`
`/`

True if the current absolute path has at least *n* elements relative to the root directory, hence `/` is counted as 0 elements.

`c`
`.`
`~`

True if the current path, with prefix replacement, has at least *n* elements relative to the root directory, hence `/` is counted as 0 elements.

`D`

True if the month is equal to *n* (January = 0).

`d`

True if the day of the month is equal to *n*.

`e`

True if the evaluation depth is at least *n*.

`g`

True if the effective gid of the current process is *n*.

`j`

True if the number of jobs is at least *n*.

`L`

True if the `SHLVL` parameter is at least *n*.

`l`

True if at least *n* characters have already been printed on the current line. When *n* is negative, true if at least `abs(`*n*`)` characters remain before the opposite margin (thus the left margin for `RPROMPT`).

`S`

True if the `SECONDS` parameter is at least *n*.

`T`

True if the time in hours is equal to *n*.

`t`

True if the time in minutes is equal to *n*.

`v`

True if the array `psvar` has at least *n* elements.

`V`

True if element *n* of the array `psvar` is set and non-empty.

`w`

True if the day of the week is equal to *n* (Sunday = 0).

`%<`*string*`<`
`%>`*string*`>`
`%[`*xstring*`]`

Specifies truncation behaviour for the remainder of the prompt string. The third, deprecated, form is equivalent to '`%`*xstringx*', i.e. *x* may be '`<`' or '`>`'. The *string* will be displayed in place of the truncated portion of any string; note this does not undergo prompt expansion.

The numeric argument, which in the third form may appear immediately after the '`[`', specifies the maximum permitted length of the various strings that can be displayed in the prompt. In the first two forms, this numeric argument may be negative, in which case the truncation length is determined by subtracting the absolute value of the numeric argument from the number of character positions remaining on the current prompt line. If this results in a zero or negative length, a length of 1 is used. In other words, a negative argument arranges that after truncation at least *n* characters remain before the right margin (left margin for `RPROMPT`).

The forms with '`<`' truncate at the left of the string, and the forms with '`>`' truncate at the right of the string. For example, if the current directory is '`/home/pike`', the prompt '`%8<..<%/`' will expand to '`..e/pike`'. In this string, the terminating character ('`<`', '`>`' or '`]`'), or in fact any character, may be quoted by a preceding '`\`'; note when using `print -P`, however, that this must be doubled as the string is also subject to standard `print` processing, in addition to any backslashes removed by a double quoted string: the worst case is therefore '`print -P "%<\\\\<<..."`'.

If the *string* is longer than the specified truncation length, it will appear in full, completely replacing the truncated string.

The part of the prompt string to be truncated runs to the end of the string, or to the end of the next enclosing group of the '`%(`' construct, or to the next truncation encountered at the same grouping level (i.e. truncations inside a '`%(`' are separate), which ever comes first. In particular, a truncation with argument zero (e.g., '`%<<`') marks the end of the range of the string to be truncated while turning off truncation from there on. For example, the prompt '`%10<...<%~%<<%# `' will print a truncated representation of the current directory, followed by a '`%`' or '`#`', followed by a space. Without the '`%<<`', those two characters would be included in the string to be truncated. Note that '`%-0<<`' is not equivalent to '`%<<`' but specifies that the prompt is truncated at the right margin.

Truncation applies only within each individual line of the prompt, as delimited by embedded newlines (if any). If the total length of any line of the prompt after truncation is greater than the terminal width, or if the part to be truncated contains embedded newlines, truncation behavior is undefined and may change in a future version of the shell. Use '`%-`*n*`(l.`*true-text*`.`*false-text*`)`' to remove parts of the prompt when the available space is less than *n*.

[<<] [>>] [Top] [Contents] [Index] [?]

This document was generated on *February 15, 2020* using *texi2html 5.0*.
Zsh version 5.8, released on February 14, 2020.