How to change the title of an xterm

Table of Contents

How to change the title of an xterm....1
Ric Lister, ric@giccs.georgetown.edu....1
1. Where to find this document....1
2. Static titles....1
3. Dynamic titles....1
4. Examples for different shells....1
5. Printing the current job name....1
6. Appendix: escapes for other terminal types....1
7. Appendix: examples in other languages....1
8. Credits....2
1. Where to find this document....2
2. Static titles....2
3. Dynamic titles....2
3.1 xterm escape sequences....2
3.2 Printing the escape sequences....3
4. Examples for different shells....3
4.1 zsh....3
4.2 tcsh....4
4.3 bash....5
4.4 ksh....5
4.5 csh....6
5. Printing the current job name....6
5.1 zsh....6
5.2 Other shells....7
6. Appendix: escapes for other terminal types....7
6.1 IBM aixterm....7
6.2 SGI wsh, xwsh and winterm....7
6.3 Sun cmdtool and shelltool....7
6.4 CDE dtterm....7
6.5 HPterm....8
7. Appendix: examples in other languages....8
7.1 C....8
7.2 Perl....8
8. Credits....9

How to change the title of an xterm

Ric Lister, `ric@giccs.georgetown.edu`

v2.0, 27 October 1999

This document explains how to use escape sequences to dynamically change window and icon titles of an xterm. Examples are given for several shells, and the appendix gives escape sequences for some other terminal types.

---

1. Where to find this document

2. Static titles

3. Dynamic titles

- 3.1 xterm escape sequences
- 3.2 Printing the escape sequences

4. Examples for different shells

- 4.1 zsh
- 4.2 tcsh
- 4.3 bash
- 4.4 ksh
- 4.5 csh

5. Printing the current job name

- 5.1 zsh
- 5.2 Other shells

6. Appendix: escapes for other terminal types

- 6.1 IBM `aixterm`
- 6.2 SGI `wsh`, `xwsh` and `winterm`
- 6.3 Sun `cmdtool` and `shelltool`
- 6.4 CDE `dtterm`
- 6.5 HPterm

7. Appendix: examples in other languages

- 7.1 C

- 7.2 Perl

8. Credits

1. Where to find this document

This document is now part of the Linux HOWTO Index and can be found at http://sunsite.unc.edu/LDP/HOWTO/mini/Xterm−Title.html.

The latest version can always be found in several formats at http://www.giccs.georgetown.edu/~ric/howto/Xterm−Title/.

This document supercedes the original howto written by Winfried Trümper.

2. Static titles

A static title may be set for any of the terminals `xterm`, `color−xterm` or `rxvt`, by using the `−T` and `−n` switches:

```
xterm −T "My XTerm's Title" −n "My XTerm's Icon Title"
```

3. Dynamic titles

Many people find it useful to set the title of a terminal to reflect dynamic information, such as the name of the host the user is logged into, the current working directory, etc.

3.1 xterm escape sequences

Window and icon titles may be changed in a running xterm by using XTerm escape sequences. The following sequences are useful in this respect:

- `ESC]0;`**`string`**`BEL` −− Set icon name and window title to **string**
- `ESC]1;`**`string`**`BEL` −− Set icon name to **string**
- `ESC]2;`**`string`**`BEL` −− Set window title to **string**

where `ESC` is the **escape** character (\033), and `BEL` is the **bell** character (\007).

Printing one of these sequences within the xterm will cause the window or icon title to be changed.

Note: these sequences apply to most xterm derivatives, such as `nxterm`, `color−xterm` and `rxvt`. Other terminal types often use different escapes; see the appendix for examples. For the full list of xterm escape sequences see the file ctlseq2.txt, which comes with the xterm distribution, or xterm.seq, which comes with the rxvt distribution.

3.2 Printing the escape sequences

For information that is constant throughout the lifetime of this shell, such as host and username, it will suffice to simply echo the escape string in the shell rc file:

```
echo -n "\033]0;${USER}@${HOST}\007"
```

should produce a title like `username@hostname`, assuming the shell variables `$USER` and `$HOST` are set correctly. The required options for `echo` may vary by shell (see examples below).

For information that may change during the shell's lifetime, such as current working directory, these escapes really need to be applied every time the prompt changes. This way the string is updated with every command you issue and can keep track of information such as current working directory, username, hostname, etc. Some shells provide special functions for this purpose, some don't and we have to insert the title sequences directly into the prompt string. This is illustrated in the next section.

---

4. Examples for different shells

Below we provide an set of examples for some of the more common shells. We start with `zsh` as it provides several facilities that make our job much easier. We will then progress through increasingly difficult examples.

In all the examples we test the environment variable `$TERM` to make sure we only apply the escapes to xterms. We test for `$TERM=xterm*`; the wildcard is because some variants (such as rxvt) can set `$TERM=xterm-color`.

We should make an extra comment about C shell derivatives, such as `tcsh` and `csh`. In C shells, undefined variables are fatal errors. Therefore, before testing the variable `$TERM`, it is necessary to test for its existence so as not to break non–interactive shells. To achieve this you must wrap the examples below in something like:

```
if ($?TERM) then
    ...
endif
```

(In our opinion this is just one of many reasons not to use C shells. See *Csh Programming Considered Harmful* for a useful discussion).

The examples below should be used by inserting them into the appropriate shell initialisation file; i.e. one that is sourced by interactive shells on startup. In most cases this is called something like `.`*`shell`*`rc` (e.g. `.zshrc`, `.tcshrc`, etc).

4.1 zsh

`zsh` provides some functions and expansions, which we will use:

```
precmd ()   a function which is executed just before each prompt
chpwd ()    a function which is executed whenever the directory is changed
\e          escape sequence for escape (ESC)
```

```
\a          escape sequence for bell (BEL)
%n          expands to $USERNAME
%m          expands to hostname up to first '.'
%~          expands to directory, replacing $HOME with '~'
```

There are many more expansions available: see the `zshmisc` man page.

Thus, the following will set the xterm title to "*username*@*hostname*: *directory*":

```
case $TERM in
    xterm*)
        precmd () {print -Pn "\e]0;%n@%m: %~\a"}
        ;;
esac
```

This could also be achieved by using `chpwd()` instead of `precmd()`. The `print` builtin works like `echo`, but gives us access to the `%` prompt escapes.

4.2 tcsh

`tcsh` has some functions and expansions similar to those of `zsh`:

```
precmd ()   a function which is executed just before each prompt
cwdcmd ()   a function which is executed whenever the directory is changed
%n          expands to username
%m          expands to hostname
%~          expands to directory, replacing $HOME with '~'
%#          expands to '>' for normal users, '#' for root users
%{...%}     includes a string as a literal escape sequence
```

Unfortunately, there is no equivalent to `zsh`'s `print` command allowing us to use prompt escapes in the title string, so the best we can do is to use shell variables (in `~/.tcshrc`):

```
switch ($TERM)
    case "xterm*":
        alias precmd 'echo -n "\033]0;${HOST}:$cwd\007"'
        breaksw
endsw
```

However, this gives the directory's full path instead of using ~. Instead you can insert the string in the prompt:

```
switch ($TERM)
    case "xterm*":
        set prompt="%{\033]0;%n@%m:%~\007%}tcsh%# "
        breaksw
    default:
        set prompt="tcsh%# "
        breaksw
endsw
```

which sets a prompt of "`tcsh% `", and an xterm title and icon of "*username*@*hostname*: *directory*". Note that the "`%{...%}`" must be placed around escape sequences (and cannot be the last item in the prompt: see the `tcsh` man page for details).

4.3 bash

`bash` supplies a variable `$PROMPT_COMMAND` which contains a command to execute before the prompt. This example sets the title to `username@hostname: directory`:

```
PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD}\007"'
```

where `\033` is the character code for `ESC`, and `\007` for `BEL`.

Note that the quoting is important here: variables are expanded in `"..."`, and not expanded in `'...'`. So `$PROMPT_COMMAND` is set to an unexpanded value, but the variables inside `"..."` are expanded when `$PROMPT_COMMAND` is used.

However, `$PWD` produces the full directory path. If we want to use the ~ shorthand we need to embed the escape string in the prompt, which allows us to take advantage of the following prompt expansions provided by the shell:

```
\u          expands to $USERNAME
\h          expands to hostname up to first '.'
\w          expands to directory, replacing $HOME with '~'
\$          expands to '$' for normal users, '#' for root
\[...\]     embeds a sequence of non-printing characters
```

Thus, the following produces a prompt of `bash$ `, and an xterm title of `username@hostname: directory`:

```
case $TERM in
    xterm*)
        PS1="\[\033]0;\u@\h: \w\007\]bash\\$ "
        ;;
    *)
        PS1="bash\\$ "
        ;;
esac
```

Note the use of `\[...\]`, which tells `bash` to ignore the non–printing control characters when calculating the width of the prompt. Otherwise line editing commands get confused while placing the cursor.

4.4 ksh

`ksh` provides little in the way of functions and expansions, so we have to insert the escape string in the prompt to have it updated dynamically. This example produces a title of `username@hostname: directory` and a prompt of `ksh$ `.

```
case $TERM in
    xterm*)
        HOST=`hostname`
        PS1='^[]0;${USER}@${HOST}: ${PWD}^Gksh$ '
        ;;
    *)
        PS1='ksh$ '
        ;;
esac
```

However, `$PWD` produces the full directory path. We can remove the prefix of `$HOME/` from the directory using the `${...##...}` construct. We can also use `${...%%...}` to truncate the hostname:

```
HOST=`hostname`
HOST=${HOST%%.*}
PS1='^[]0;${USER}@${HOST}: ${PWD##${HOME}/}^Gksh$ '
```

Note that the `^[` and `^G` in the prompt string are single characters for `ESC` and `BEL` (can be entered in emacs using `C-q ESC` and `C-q C-g`).

4.5 csh

This is very difficult indeed in `csh`, and we end up doing something like the following:

```
switch ($TERM)
    case "xterm*":
        set host=`hostname`
        alias cd 'cd \!*; echo -n "^[]0;${user}@${host}: ${cwd}^Gcsh% "'
        breaksw
    default:
        set prompt='csh% '
        breaksw
endsw
```

where we have had to alias the `cd` command to do the work of sending the escape sequence. Note that the `^[` and `^G` in the string are single characters for `ESC` and `BEL` (can be entered in emacs using `C-q ESC` and `C-q C-g`).

Notes: on some systems `hostname -s` may be used to get a short, rather than fully-qualified, hostname. Some users with symlinked directories may find `` `pwd` `` (backquotes to run the `pwd` command) gives a more accurate path than `$cwd`.

---

5. Printing the current job name

Often a user will start a long-lived foreground job such as `top`, an editor, an email client, etc, and wishes the name of the job to be shown in the title. This is a more thorny problem and is only achieved easily in `zsh`.

5.1 zsh

`zsh` provides an ideal builtin function for this purpose:

```
preexec()   a function which is just before a command is executed
$*,$1,...   arguments passed to preexec()
```

Thus, we can insert the job name in the title as follows:

```
case $TERM in
    xterm*)
      preexec () {
        print -Pn "\e]0;$*\a"
      }
    ;;
```

```
esac
```

Note: the `preexec()` function appeared around version 3.1.2 of `zsh`, so you may have to upgrade from an earlier version.

5.2 Other shells

This is not easy in other shells which lack an equivalent of the `preexec()` function. If anyone has examples please email them to the author.

---

6. Appendix: escapes for other terminal types

Many modern terminals are descended from `xterm` or `rxvt` and support the escape sequences we have used so far. Some proprietary terminals shipped with various flavours of unix use their own escape sequences.

6.1 IBM `aixterm`

`aixterm` recognises the `xterm escape` sequences.

6.2 SGI `wsh`, `xwsh` and `winterm`

These terminals set `$TERM=iris-ansi` and use the following escapes:

- `ESCP1.y`*`string`*`ESC\  Set window title to `*`string`*
- `ESCP3.y`*`string`*`ESC\  Set icon title to `*`string`*

For the full list of `xwsh` escapes see the `xwsh(1G)` man page.

The Irix terminals also support the `xterm` escapes to individually set window title and icon title, but not the escape to set both.

6.3 Sun `cmdtool` and `shelltool`

`cmdtool` and `shelltool` both set `$TERM=sun-cmd` and use the following escapes:

- `ESC]l`*`string`*`ESC\  Set window title to `*`string`*
- `ESC]L`*`string`*`ESC\  Set icon title to `*`string`*

These are truly awful programs: use something else.

6.4 CDE `dtterm`

`dtterm` sets `$TERM=dtterm`, and appears to recognise both the standard `xterm` escape sequences and the Sun `cmdtool` sequences (tested on Solaris 2.5.1, Digital Unix 4.0, HP−UX 10.20).

6.5 HPterm

`hpterm` sets `$TERM=hpterm` and uses the following escapes:

- `ESC&f0k`*`length`*`D`*`string`* Set window title to *string* of length *length*
- `ESC&f-1k`*`length`*`D`*`string`* Set icon title to *string* of length *length*

A basic C program to calculate the length and echo the string looks like this:

```
#include <string.h>
int main(int argc, char *argv[])
{
    printf("\033&f0k%dD%s", strlen(argv[1]), argv[1]);
    printf("\033&f-1k%dD%s", strlen(argv[1]), argv[1]);
    return(0);
}
```

We may write a similar shell–script, using the `${#string}` (`zsh`, `bash`, `ksh`) or `${%string}` `(tcsh)` expansion to find the string length. The following is for `zsh`:

```
case $TERM in
    hpterm)
        str="\e]0;%n@%m: %~\a"
        precmd () {print -Pn "\e&f0k${#str}D${str}"}
        precmd () {print -Pn "\e&f-1k${#str}D${str}"}
        ;;
esac
```

7. Appendix: examples in other languages

It may be useful to write a small program to print an argument to the title using the `xterm` escapes. Some examples are provided below.

7.1 C

```
#include <stdio.h>

int main (int argc, char *argv[]) {
  printf("%c]0;%s%c", '\033', argv[1], '\007');
  return(0);
}
```

7.2 Perl

```
#!/usr/bin/perl
print "\033]0;@ARGV\007";
```

8. Credits

Thanks to the following people who have provided advice, errata, and examples for this document.

Paul D. Smith <psmith@BayNetworks.COM> and Christophe Martin <cmartin@ipnl.in2p3.fr> both pointed out that I had the quotes the wrong way round in the bash $PROMPT_COMMAND. Getting them right means variables *are* expanded dynamically.

Paul D. Smith <psmith@BayNetworks.COM> suggested the use of \[...\] in the bash prompt for embedding non–printing characters.

Christophe Martin <cmartin@ipnl.in2p3.fr> provided the solution for ksh.

Keith Turner <keith@silvaco.com> supplied the escape sequences for Sun cmdtool and shelltool.

Jean–Albert Ferrez <ferrez@dma.epfl.ch> pointed out some inconsistencies in the use of "PWD" and "$PWD", and in the use of "\" vs "\\".

Bob Ellison <papillo@hpellis.fc.hp.com> and Jim Searle <jims@broadcom.com> tested dtterm on HP–UX.

Teng–Fong Seak <seak@drfc.cad.cea.fr> suggested the –s option for hostname, use of `pwd`, and use of echo under csh.

Trilia <trilia@nmia.com> suggested examples in other languages.

Brian Miller <bmiller@telstra.com.au> supplied the escape sequences and examples for hpterm.

Lenny Mastrototaro <lenny@click3x.com> explained the Irix terminals' use of xterm escape sequences.

Paolo Supino <paolo@init.co.il> suggested the use of \\$ in the bash prompt.