# *Learning_zsh_shell*

Reference_book: shell_scripting_zsh

Reference_Online Source: Online Source

# zsh-shell

## History of zsh shell

-Open source unix shell developed by Paul Falstad.

Some of the features that make the zsh shell unique are:

- Improved shell option handling
- Shell compatibility modes
- Loadable modules

The benefit of built-in commands is execution speed.

## Parts of the zsh shell

Upcoming built-in cmds, cli parameters and env variables.

### Shell options

Most shells use command line parameters to define the behavior of the shell. The zsh shell uses a few command line parameters to define the operation of the shell, but mostly it uses *options* to customize the behavior of the shell. You can set shell options either on the command line, or within the shell itself using the `set` command.

By far the zsh shell is the most customizable shell available.

**The zsh Shell Command Line Parameters**

| Parameter | Description |
|---|---|
| `-c` | Execute only the specified command and exit. |
| `-i` | Start as an interactive shell, providing a command line interface prompt. |
| `-s` | Force the shell to read commands from `STDIN`. |
| `-o` | Specify command line options. |

## Shell state

There are six different zsh shell options that define the type of shell to start:

- interactive (`-i`): Provides a command line interface prompt for entering built-in commands and program names.
- login (`-l`): The default zsh shell type, processes the zsh shell startup files, and provides a command line interface prompt.
- privileged (`-p`): The default if the effective user ID (UID) of the user is not the same as the real UID (the user has become the root user). This option disables the user startup files.
- restricted (`-r`): Restricts the user to a specified directory structure in the shell.
- shin_stdin (`-s`): Commands are read from `STDIN`.
- single_command (`-t`): Executes a single command from `STDIN` and exits.

The shell states define whether or not the shell starts with a command line interface prompt, and what access the user has within the shell.

## The zshell files

-The zsh shell uses files at both login and logout to allow you to preset variables and features of the zsh shell.

-automatically looks for default files for setting environment.

There are 4 types of zsh file that shell can handle.

1. Shell startup file
2. login file
3. Interactive file
4. logout file

### *1. The shell startup files*

When you start a new zsh shell as a login shell (either by logging into the system, or by running the zsh shell program), the zsh shell looks for commands in two files.

The default system-wide zsh shell startup file is:

/etc/zshenv

After executing this file (if it exists), the zsh shell proceeds to the user's $HOME directory and looks for the file:

$HOME/.zshenv

You can test the order in which the zsh shell executes these files by performing a simple test using the echo statement in each file:

```
WARNING: The locate database (/var/db/locate.database) does not exist.
To create the database, run the following command:

  sudo launchctl load -w /System/Library/LaunchDaemons/com.apple.locate.plist

Please be aware that the database can take some time to generate; once
the database has been created, this message will no longer appear.
```

## *2. The shell login files*

If u use the zsh shell as a login shell, it first looks for and executes two files.

- /etc/zlogin
- /etc/zprofile

These files are executed for all users who use the zsh shell as their login shell. The files should contain system-wide **environment variables** that should be set for all users, and executable pro- grams that all users should use (such as the **umask** command to set default file and directory **permissions**). Most likely if you're using a Linux distribution that supports the zsh shell, these files already exist for setting system environment variables at login.

After executing this file, the zsh shell looks for two files in each user's $HOME directory:

- $HOME/.zlogin
- $HOME/.zprofile

These files can contain user-specific environment variable settings, and executable commands that an individual user wants to run when logging in to the system, **before the command line interface prompt appears.**

Any environment variable settings you make in the **$HOME/.zprofile** shell will **override** any system-wide settings made in the /etc/**zprofile** file, and similarly for the **.zlogin** files.

```
% zsh -l
This is the /etc/zshenv file.
This is the .zshenv file in HOME.
This is the /etc/zprofie file.
This is the .zprofile file in HOME.
This is the /etc/zlogin file.
This is the .zlogin file in HOME.
```

### 3. The interactive shell files

If you start an interactive zsh shell session, there's another set of files that can hold startup variables and commands:

- /etc/zshrc
- $HOME/.zshrc

```
% zsh -l
This is the /etc/zshenv file.
This is the .zshenv file in HOME.
This is the /etc/zprofie file.
This is the .zprofile file in HOME.
This is the /etc/zshrc file.
This is the .zshrc file in HOME.
This is the /etc/zlogin file.
This is the .zlogin file in HOME.
%
```

Notice the order in which the zsh shell executes the files. The zshrc file pair are executed after the zprofile files, but before the zlogin files.

### 4. The shell logout files

Besides the login and startup files, the zsh shell also has the ability to execute commands in files when you log out from an interactive or login shell session. These files can be in the following locations:

- /etc/zlogout
- $HOME/.zlogout

{As you probably figured out by now, every user on the system executes the commands in the /etc/zlogout file at logout, while each individual user has a unique $HOME/.zlogout file. The zsh shell will execute any commands in these files before logging out of the current shell. } its what the name says

{This doesn't include just exiting from an interactive shell. Thus, if you just start another zsh shell from an existing shell session, the zsh shell won't execute the logout files when you exit the shell. However, if you start a login shell, the logout files should execute upon exiting: } confusing

```
% zsh -l
This is the /etc/zshenv file.
This is the .zshenv file in HOME.
This is the /etc/zprofie file.
This is the .zprofile file in HOME.
This is the /etc/zshrc file.
This is the .zshrc file in HOME.
This is the /etc/zlogin file.
This is the .zlogin file in HOME.
% exit
This is the .zlogout file in HOME.
This is the /etc/zlogout file.
%
```

Notice that the shell executes the `.zlogout` file before the global `zlogout` file, which is in the opposite order from the other zsh file types.

There are five startup files that zsh will read commands from:

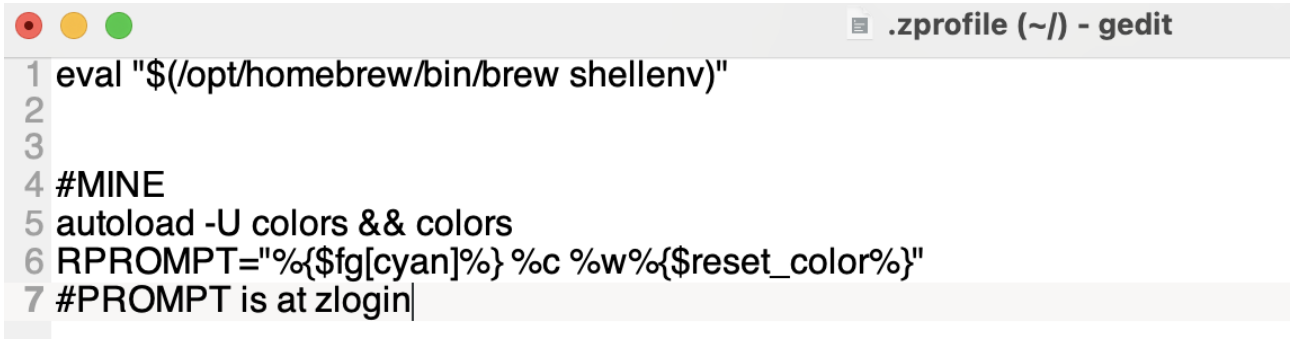
```
$ZDOTDIR/.zshenv
$ZDOTDIR/.zprofile
$ZDOTDIR/.zshrc
$ZDOTDIR/.zlogin
$ZDOTDIR/.zlogout
```

If ZDOTDIR is not set, then the value of HOME is used; this is the usual case.

`.zshenv' is sourced on all invocations of the shell, unless the -f option is set. It should contain commands to set the command search path, plus other important environment variables. `.zshenv' should not contain commands that produce output or assume the shell is attached to a tty.

`.zshrc' is sourced in interactive shells. It should contain commands to set up aliases, functions, options, key bindings, etc.

`.zlogin' is sourced in login shells. It should contain commands that should be executed only in login shells. `.zlogout' is sourced when login shells exit. `.zprofile' is similar to `.zlogin', except that it is sourced before `.zshrc'. `.zprofile' is meant as an alternative to `.zlogin' for ksh fans; the two are not intended to be used together, although this could certainly be done if desired. `.zlogin' is not the place for alias definitions, options, environment variable settings, etc.; as a general rule, it should not change the shell environment at all. Rather, it should be used to set the terminal type and run a series of external commands (fortune, msgs, etc).





I think it takes few secs. So its better for zprofile to load my func into memory.

The zsh environment variables:

| | |
|---|---|
| `fpath` | An array of directories specifying the search path for functions |
| `FPATH` | The same as `fpath`, but using a colon as the field separator. |
| `PROMPT` | The primary prompt string used by the shell |
| `PROMPT2` | The same as `PROMPT` |
| `PS1` | The same as `PROMPT` |
| `PS2` | The same as `PROMPT2` |
| `psvar` | An array whose first nine values are referenced by the associated prompt strings. |
| `PSVAR` | The same as `psvar`, but using a colon as the field separator. |
| `RANDOM` | A random number generator for integers between 0 and 32767. |
| `RPROMPT` | A prompt displayed on the right side of the command line interface. |
| `RPS1` | The same as `RPROMPT`. |
| `RPROMPT2` | A prompt displayed on the right side of the command line interface when more input is required for a command. |
| `RPS2` | The same as `RPROMPT2`. |
| `reply` | Reserved for passing string values between shell scripts and functions. |
| `REPLY` | The same as reply, but uses an array value rather than string values. |
| `ZDOTDIR` | The directory to search for zsh personal startup files. The default is the `$HOME` directory. |

## Typeset command

The zsh shell supports the typeset command, which allows you to declare attributes for a vari- able before using it. Table 23-3 show the options available for the zsh typeset command.

```
kazimsyed@kazims-MacBook-Air scripts % typeset -A rating
kazimsyed@kazims-MacBook-Air scripts % rating=(DeathNote 9 Code_Geass 8 HunterxHunter 9)
kazimsyed@kazims-MacBook-Air scripts % echo ${rating[DeathNote]}
9
```

```
echo "Shell started $$"

#creating a numeric array
typeset -a ginti
ginti=(7 8 6  1 1)

for e in ${ginti[*]}
do
echo $e
done
```

```
kazimsyed@kazims-MacBook-Air scripts % typeset | grep -e "whoru" -e "special"
array special=( 1 7 8 6 )
whoru
kazimsyed@kazims-MacBook-Air scripts % echo $whoru
kaju
```

*Figure 1: -h whoru : to create a variable whose value is hidden*

```
kazimsyed@kazims-MacBook-Air scripts % typeset -r apple=faraaz
kazimsyed@kazims-MacBook-Air scripts % typeset | grep -e apple
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.rK5WznfSsK/Listeners
VENDOR=apple
__CFBundleIdentifier=com.apple.Terminal
readonly apple=faraaz
kazimsyed@kazims-MacBook-Air scripts % apple=mango
zsh: read-only variable: apple
kazimsyed@kazims-MacBook-Air scripts % typeset | grep -e apple
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.rK5WznfSsK/Listeners
VENDOR=apple
__CFBundleIdentifier=com.apple.Terminal
readonly apple=faraaz
```

*Figure 2: using readonly variables*

## The typeset Command Parameters

| Parameter | Description |
|---|---|
| -a | Create a numerical array variable. |
| -A | Create an associative array variable. |
| -E | Create a double-precision floating-point variable and display using scientific notation. |
| -f | Define a function name instead of a variable. |
| -F | Create a double-precision floating-point variable and display using fixed-point decimal notation. |
| -h | Create a hidden special variable. |
| -H | Don't display the value of the variable. |
| -i | Create an integer data type variable. |
| -l | Convert the variable value to lower case. |
| -L | Left-justify by removing leading blanks from a variable. |
| -r | Make the specified variables read-only. |
| -R | Right-justify by adding blanks on the left. |
| -t | Tag the specified variables. |
| -u | Convert the variable value to upper case. |
| -U | Keep only the first occurrence for each duplicated value in a numerical array. |
| -x | Mark the specified variable for automatic export. |
| -Z | Right-justify using leading zeroes. |

## Built-in commands

The zsh shell is unique in that it allows you to expand the built-in commands available in the shell.

### The zsh core built-in commands:

Index:

## Filename Generation

| Commands | Working |
|---|---|
| Ls *.[co] | Files ending with c or o |
| Ls foo.? | Files ending with single letter |
| Ls *.[^c] | Files not ending with c |
| | |

Also, if the EXTENDEDGLOB option is set, some new features are activated. For example, the^ character negates the pattern following it:

% setopt extendedglob

| Commands | Working |
|---|---|
| `Ls -d ^.*c` | `Not ending with c` |
| `Ls -d ^*.*` | `Not having an extension` |
| `Ls -d ^*.py` | `No python files are desired` |

```
kazimsyed@kazims-MacBook-Air shell_script % ls *[0-9][0-9][0-9]
temp275
kazimsyed@kazims-MacBook-Air shell_script % ls *[200-300]
zsh: no matches found: *[200-300]
```

An expression of the form<x-y>matches a range of integers:

| Commands | Working |
|---|---|
| ls run<200-300> | Run235 |
| Ls run<-200> | run2 |
| | |

Grouping

| | |
|---|---|
| *.(py\|sh) | |

It is possible to exclude certain files from the patterns using the~character. A pattern of the form*.c~bar.c lists all files matching*.c, except for the file`bar.c'.

```
% ls *.c
foo.c    foob.c    bar.c
% ls *.c~bar.c
foo.c    foob.c
% ls *.c~f*
bar.c
```

```
kazimsyed@kazims-MacBook-Air Desktop % ls -Fd *(\/)
Node/           shell_script/
```

One can add a number of *qualifiers* to the end of any of these patterns, to restrict matches to certain file types. A qualified pattern is of the form.

```
kazimsyed@kazims-MacBook-Air Desktop % ls -F *(@)
Documents Part2@
```

| Commands | Working |
|---|---|
| Ls -Fd *( \/ ) | To find all directories. Here -d is used, so it doesn't expand. |
| Ls -F *(@) | Symbolic Link |

Note that *(x) and *(*) both match executables. *(X) matches files executable by others, as opposed to *(x), which matches files executable by the owner. *(R) and *(r) match readable files; *(W) and *(w), which checks for writable files. *(W) is especially important, since it checks for world-writable files:

| Commands | Working |
|---|---|
| Ls -F *(x) or Ls -F *(*) | Both match executables |
| Ls -F *(X) | This match files executable by owner |
| Ls -F *(R) or Ls -F(r) | readables |
| Similaryly *(w) | writable |

*(U) matches all files owned by you. To search for all files not owned by you, use *(^U):

```
% l -l *(^U)
-rw-------  1 subbarao       29 May 23 18:13 sub
```

## Startup Files

There are 5 startup files that zsh will read commands from:

```
$ZDOTDIR/.zshenv
$ZDOTDIR/.zprofile
$ZDOTDIR/.zshrc
$ZDOTDIR/.zlogin
$ZDOTDIR/.zlogout
```

skip

```
==> Next steps:
- Run these two commands in your terminal to add Homebrew to your PATH:
    echo 'eval "$(/opt/homebrew/bin/brew shellenv)"' >> /Users/kazimsyed/.zprofile
    eval "$(/opt/homebrew/bin/brew shellenv)"
- Run brew help to get started
- Further documentation:
    https://docs.brew.sh
```

## Shell functions

```
% randline () {
>       integer z=$(wc -l <$1)
>       sed -n $[RANDOM % z + 1]p $1
> }
% randline /etc/motd
```

Here's another one:

```
% cx () { chmod +x $* }
% ls -l foo bar
-rw-r--r--  1 pfalstad       29 May 24 04:38 bar
-rw-r--r--  1 pfalstad       29 May 24 04:38 foo
% cx foo bar
% ls -l foo bar
-rwxr-xr-x  1 pfalstad       29 May 24 04:38 bar
-rwxr-xr-x  1 pfalstad       29 May 24 04:38 foo
```

Note that this could also have been implemented as an alias:

```
% chmod 644 foo bar
% alias cx='chmod +x'
% cx foo bar
% ls -l foo bar
-rwxr-xr-x  1 pfalstad       29 May 24 04:38 bar
-rwxr-xr-x  1 pfalstad       29 May 24 04:38 foo
```

```
kazimsyed@kazims-MacBook-Air scripts % zsh -l
kazimsyed@kazims-MacBook-Air scripts % fpath[4]=/tmp/funs
kazimsyed@kazims-MacBook-Air scripts % ls /tmp/funs
shoot           trash_shoot
kazimsyed@kazims-MacBook-Air scripts % autoload shoot
kazimsyed@kazims-MacBook-Air scripts % mktemp testing.XX
testing.gr
kazimsyed@kazims-MacBook-Air scripts % shoot tes*
shell startedd at 2721
trash exist
successfully send testing.gr.Sep-2022-02-08.R0 file to trash_shoot
kazimsyed@kazims-MacBook-Air scripts % ls
1.sh            2378.12         4.sh            6.sh            data2           shoot           trash_shoot
2.sh            3.sh            5.sh            data1           output          template
kazimsyed@kazims-MacBook-Air scripts % ls trash_shoot; sort -r
data21.Sep-2022-01-18.jq        regex_check.Sep-2022-00-41.Us   esting.gr.Sep-2022-02-08.R0
```

*Figure 3: How to use functions and **autoload** cmd*

```
#!/bin/bash



function shoot
{
echo "shell startedd at $$"
if [ -d "trash_shoot" ]
then
  echo "trash exist"
else
   `mkdir trash_shoot`
    echo "empty trash_shoot successfully created"
fi

   for e in $@
   do
      name="$e.`date +%h-%Y-%H-%M`"
      name=`mktemp ${name}.XX`
      `mv $e $name`
      `mv $name ./trash_shoot`
      success=$?
      if [[ success -eq 0 ]]
       then
          echo successfully send  $name file to trash_shoot
      fi
   done

}

shoot $@

```

Another way:

```
% echo $fpath
/usr/local/share/zsh/site-functions
/usr/share/zsh/4.2.5/functions/Completion
/usr/share/zsh/4.2.5/functions/Completion/AIX
/usr/share/zsh/4.2.5/functions/Completion/BSD
```

```
% cat dbl
#!/bin/zsh
# a function to double a value
dbl() {
   value=$(( $1 * 2 ))
   return $value
}
% cp dbl /usr/local/share/zsh/site-functions
%
```

*Figure 4: this will make permanent by copying it to a directory.*

## Directory stacks:

```
kazimsyed@kazims-MacBook-Air scripts % pushd
~ ~/Desktop/shell_script/scripts
kazimsyed@kazims-MacBook-Air ~ % ls
Applications    Downloads       Music           file.txt.save
Desktop         Library         Pictures        mydesktop
Documents       Movies          Public
kazimsyed@kazims-MacBook-Air ~ % cd Downloads
kazimsyed@kazims-MacBook-Air Downloads % cd ..
kazimsyed@kazims-MacBook-Air ~ % cd Documents
kazimsyed@kazims-MacBook-Air Documents % popd
~/Desktop/shell_script/scripts
kazimsyed@kazims-MacBook-Air scripts %
```

*Figure 5: Working of pushd and popd*

```
kazimsyed@kazims-MacBook-Air ~ % cd Documents
kazimsyed@kazims-MacBook-Air Documents % dirs -v
0       ~/Documents
1       ~/Desktop/shell_script/scripts
2       ~/Downloads/E
kazimsyed@kazims-MacBook-Air Documents % pushd ~
~ ~/Documents ~/Desktop/shell_script/scripts ~/Downloads/E
kazimsyed@kazims-MacBook-Air ~ % dirs -v
0       ~
1       ~/Documents
2       ~/Desktop/shell_script/scripts
3       ~/Downloads/E
kazimsyed@kazims-MacBook-Air ~ % cd +3
~/Downloads/E
kazimsyed@kazims-MacBook-Air E % dirs -v
0       ~/Downloads/E
1       ~/Documents
2       ~/Desktop/shell_script/scripts
```

*Figure 6: instead of cd, we can use pushd, which will push the pwd and then cd to !:1*

```
~> DIRSTACKSIZE=8
~> setopt autopushd pushdminus pushdsilent pushdtohome
~> alias dh='dirs -v'
~> cd /tmp
```

pushdminus not necessary.

Pushdsilent will not show the directory stack.

Autopushd will make cd to behave like pushd.

Directory stack works like this.

| Present | Modern | Medieval | Older | Oldest |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

present is not included by pushd, while the rest are.

Use **dirs -v** cmd

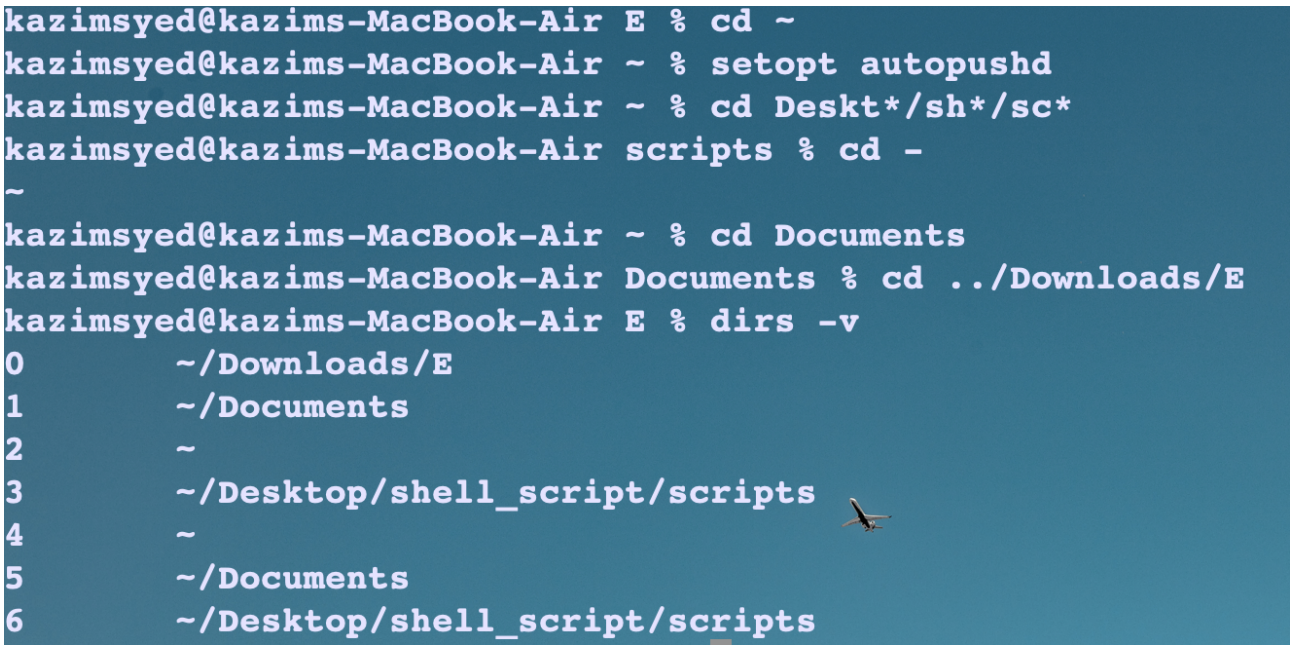



*Figure 7: setopt autopushd*

## Command substitution

Command substitution in zsh can take two forms. In the traditional form, a command enclosed in backquotes (`...`) is replaced on the command line with its output. This is the form used by the older shells. Newer shells (like zsh) also provide another form, $(...). This form is much easier to nest.

| Old shell | ` ... ` | New shell | $(...) |
|---|---|---|---|
| `echo Todays Date:`date +%d-%m-%y`` | | `echo Todays Date:$(date +%d-%m-%y)` | |

--

```
kazimsyed@kazims-MacBook-Air Downloads % print -l kazim syed
kazim
syed
```

*Figure 8: print -l : prints its arguments one per line.*

```
kazimsyed@kazims-MacBook-Air scripts % grep -e nobody -e daemon data1
nobody                  *  -2
daemon                  *  1
```

*Figure 9: Use of grep -e command for search*

```
% alias fm='finger -m'
% fm root
Login name: root                        In real life: Operator
Directory: /                            Shell: /bin/csh
On since May 19 10:41:15 on console     3 days 5 hours Idle Time
No unread mail
No Plan.
```

```
kazimsyed@kazims-MacBook-Air scripts % finger
Login    Name                TTY   Idle   Login  Time   Office  Phone
kazimsyed kazim syed         *con     3d  Fri    14:18
kazimsyed kazim syed          s00         Mon    14:20
```

*Table 1: use of remote command to learn about Remote users*

```
[kazimsyed@kazims-MacBook-Air scripts % ls -F *(R)
1.sh*           data1           template
[kazimsyed@kazims-MacBook-Air scripts % ls -F *(r)
1.sh*   data1
[kazimsyed@kazims-MacBook-Air scripts % ls -l
total 24
-rwxr--r--@ 1 kazimsyed  staff  529 Aug 28 17:20 1.sh
-rw-r--r--  1 kazimsyed  staff  297 Aug 28 15:09 data1
----r--r--  1 kazimsyed  staff   36 Aug 28 16:09 template
```

*Figure 10: *(r) readable by me*

```
% rm () { mv $* /tmp/wastebasket }
% rm foo.dvi
% ls /tmp/wastebasket
foo.dvi
```

*Figure 11: Making rm safe by using a bit bucket*

--

## History:

typing fc will execute an editor on this command, allowing you to fix it. (The default editor is vi, by the way, not ed).

## Binding

```
Last login: Fri Sep  2 03:29:19 on ttys000
[kazimsyed@kazims-MacBook-Air ~ % bindkey -s '^' 'uptime'
[kazimsyed@kazims-MacBook-Air ~ % uptime
 3:31  up 13:02, 2 users, load averages: 1.26 1.31 1.34
```

*Figure 12: creating shortcuts, I have used just caret instead of caret and T.*

The -s flag to bindkey specifies that you are binding the key to a string, not a command. Thus bindkey -s '^T' 'uptime\n' lets you VMS lovers get the load average whenever you press ^T.

Another use of the editor is to edit the value of variables. For example, an easy way to change your path is to use the `vared` command:

```
% vared PATH
> /u/pfalstad/scr:/u/pfalstad/bin/sun4:/u/maruchck/scr:/u/subbarao/bin:/u/maruc
hck/bin:/u/subbarao/scripts:/usr/princeton/bin:/usr/ucb:/usr/bin:/bin:/usr/host
s:/usr/princeton/bin/X11:/./usr/lang:/./usr/etc:/./etc
```

You can now edit the path. When you press return, the contents of the edit buffer will be assigned to `PATH`.

*Figure 13: Be cautius*

## Parameter substitution

```
kazimsyed@kazims-MacBook-Air scripts % x=kazim
kazimsyed@kazims-MacBook-Air scripts % echo $x:u
KAZIM
kazimsyed@kazims-MacBook-Air scripts % echo $x:u:l
kazim
```

## Shell parameter

In general, parameters with names in all lowercase are arrays; assignments to them take the form:

name=(elem ...)

Parameters with names in all uppercase are strings. If there is both an array and a string version of the same parameter, the string version is a colon-separated list, like PATH.

```
kazimsyed@kazims-MacBook-Air scripts % echo $path
/opt/homebrew/bin /opt/homebrew/sbin /usr/local/bin /usr/bin /bin /usr/sbin /sbin
kazimsyed@kazims-MacBook-Air scripts % echo ${path[1]}
/opt/homebrew/bin
kazimsyed@kazims-MacBook-Air scripts % echo ${path[2]}
/opt/homebrew/sbin
kazimsyed@kazims-MacBook-Air scripts % echo ${path[3]}
/usr/local/bin
kazimsyed@kazims-MacBook-Air scripts % echo ${path[4]}
/usr/bin
kazimsyed@kazims-MacBook-Air scripts % echo ${path[*]}
/opt/homebrew/bin /opt/homebrew/sbin /usr/local/bin /usr/bin /bin /usr/sbin /sbin
```

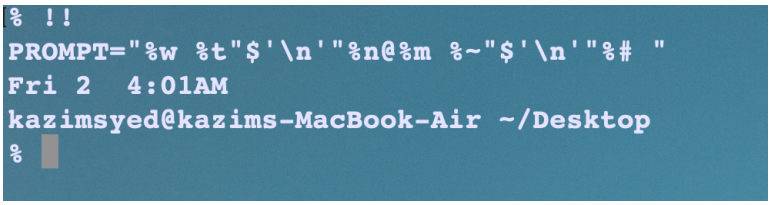
## Prompting

| Parameter | working | about |
|---|---|---|
| %m | hostname | |
| %# | Stands for # or % | Depending on the shell whether its running a root user or not |
| %h | No of current history events | A number |
| %t | Clock | 12:00am/pm format |
| %w | Day and Date | |
| %n | Name of the user | |
| %~ | pwd(relative path) | |
| %c | Absolute path | |

The POSTEDIT parameter is printed whenever the editor exits. This can be useful for termcap tricks. To highlight the prompt and command line while leaving command output unhighlighted, try this:

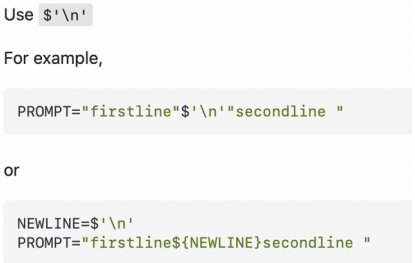
```
% POSTEDIT=`echotc se`
% PROMPT='%S%% '
```

Usecase:

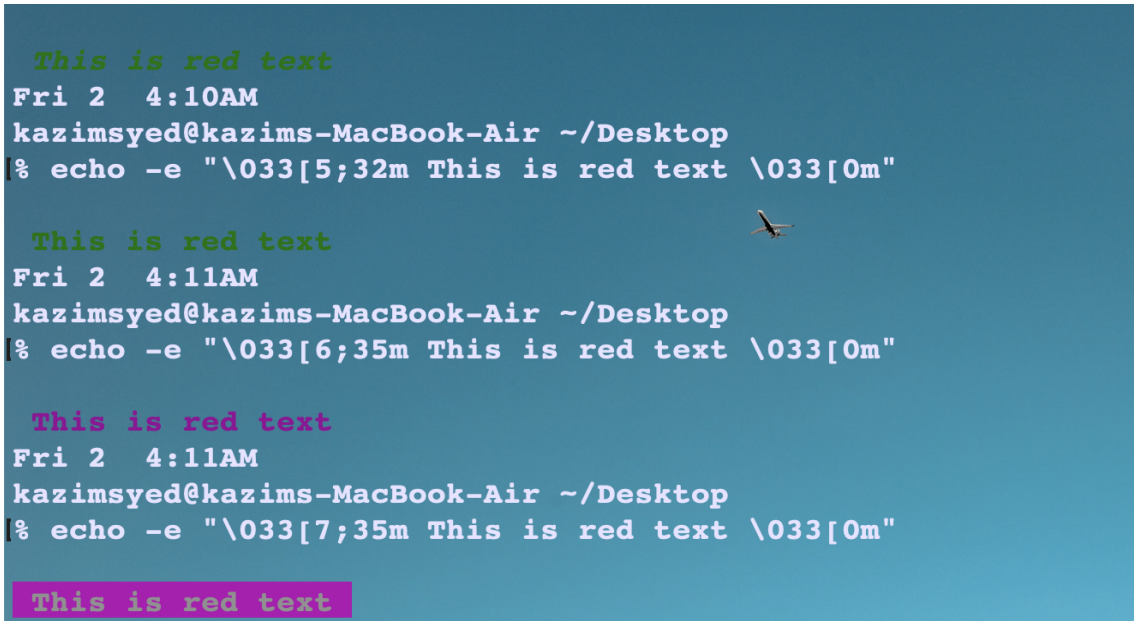



PROMPT="%w %t"$'\n'"%n@%m %~"$'\n'"%# "

link:



Adding color:



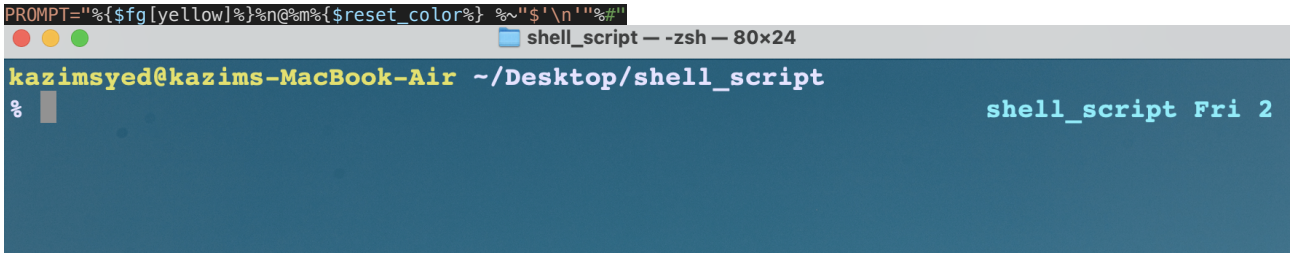

3: tells foreground e.g 35 tells magenta text
4: tells background
just 7: reverse foreground and background colors, blink fast, underline, bold,.. extra.

---

```
PS1="%{$fg[red]%}%n%{$reset_color%}@%{$fg[blue]%}%m %{$fg[yellow]%}%(5~|%-1~/.../%3~|%4~) %{$reset_color%}%% "

PROMPT=$"%{$fg[cyan]%}%w %t%{$reset_color%}"$'\n'"%{$fg[yellow]%}%n@%m%{$reset_color%} %~"$'\n'"%# "
"""

RPROMPT="%{$fg[cyan]%} %c %w%{$reset_color%}"
PROMPT="%{$fg[yellow]%}%n@%m%{$reset_color%} %~"$'\n'"%#"
```

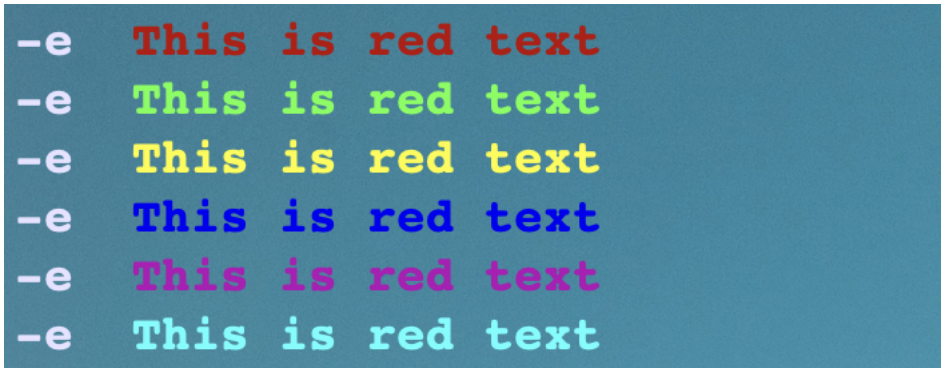


## The ANSI Color Control Codes

| Code | Description |
|---|---|
| 0 | Black |
| 1 | Red |
| 2 | Green |
| 3 | Yellow |
| 4 | Blue |
| 5 | Magenta |
| 6 | Cyan |
| 7 | White |

## The ANSI SGR Effect Control Codes

| Code | Description |
|---|---|
| 0 | Reset to normal mode. |
| 1 | Set to bold intensity. |
| 2 | Set to faint intensity. |
| 3 | Use italic font. |
| 4 | Use single underline. |
| 5 | Use slow blink. |
| 6 | Use fast blink. |
| 7 | Reverse foreground/background colors. |
| 8 | Set foreground color to background color (invisible text). |



## Login / Logout watching

You can specify login or logout events to monitor by setting the `watch` variable. Normally, this is done by specifying a list of usernames.

```
% watch=( pfalstad subbarao sukthnkr egsirer )
```

The `log` command reports all people logged in that you are watching for.

```
% log
pfalstad has logged on p0 from mickey.
pfalstad has logged on p5 from mickey.
```

If you specify hostnames with an `@` prepended, the shell will watch for all users logging in from the specified host.

```
% watch=( @mickey @phoenix )
% log
djthongs has logged on q2 from phoenix.
pfalstad has logged on p0 from mickey.
```

If you have a `.friends' file in your home directory, a convenient way to make zsh watch for all your friends is to do this:

```
% watch=( $(< ~/.friends) )
% echo $watch
subbarao maruchck root sukthnkr ...
```

If `watch` is set to `all`, then all users logging in or out will be reported.