NAME

zsh−lovers − tips, tricks and examples for the Z shell

OVERVIEW

Whenever we look at the zsh manual we wonder why there are no examples for those simply things in (shell) life. The zsh contains many features, but there was no manpage with some examples (like procmailex(5)). That's why we wrote this manpage.

Most of the tricks and oneliner come from the mailinglists zsh−users, zsh−workers, google, newsgroups and from ourself. See section **LINKS** for details.

Note: This manpage (zsh-lovers(1)) is **not** an offical part of the Z shell! It's just a just for fun − manpage ;) For comments, bugreports and feedback take a quick look at the section **BUGS.**

EXAMPLES

REDIRECTION

See also *man 1 zshmisc*.

```
null command shorthands:
 "< file" is like "$READNULLCMD <file"
 "> file" is like "cat >file"
 ">> file" is like "cat >>file"


Append ‘exit 1’ at the end of all *.sh − files:
 $ echo "exit 1" >> *.sh


Append /etc/services at the end of file ‘foo’ and ‘bar’:
 $ cat /etc/services >> foo >> bar


Pipe STDERR:
 $ echo An error >&2 2>&1 | sed -e ’s/A/I/’
```

MULTIPLE I/O REDIRECTION

Requires *setopt multios*! Some examples:

```
Print output of ’ls’ into files ’foo’ and ’bar’:
 $ ls >foo >bar


Send standard output of one process to standard input of several processes in the pipeline:
 $ process1 > >(process1) > >(process2)


Redirection to file as well as send on to pipe:
 $ make install > /tmp/logfile | grep -i error


Redirect stderr to a command like xless without redirecting stdout as well:
 $ foo 2>>(xless)
... but this executes the command asynchronously. To do it synchronously:
 $ { { foo 1>&3 } 2>&1 | xless } 3>&1
```

Redirect stderr two times:

```
 $ setopt multios ; program 2> file2 > file1 2>&1
```

More fun with stderr:

```
 $ ./my-script.sh 2> >(grep -v geek >error.log) | process-output > output.log
   echo "Thats STDOUT" >>(sed 's/stdout/another example/' > foobar)
```

MODIFIERS USAGE

Modifiers are a powerful mechanism that lets you modify the results returned by parameter, filename and history expansion. See zshexpn(1) for details.

Remove a trailing pathname component, leaving the head. This works like ‘dirname’:

```
  $ echo =ls(:h)
  /bin
```

Remove all leading pathname components, leaving the tail. This works like ‘basename’.

```
  $ echo =ls(:t)
  ls
```

Remove a filename extension of the form ‘.xxx’, leaving the root name.

```
  $ echo $PWD
  /usr/src/linux
  $ echo $PWD:t
  linux
```

Remove all but the extension.

```
  $ foo=23.42
  $ echo $foo
  23.42
  $ echo $foo:e
  42
```

Print the new command but do not execute it. Only works with history expansion.

```
  $ echo =ls(:h)
  /bin
  $ !echo:p
  $ echo =ls(:h)
```

Quote the substituted words, escaping further substitutions.

```
  $ bar="23'42"
  $ echo $bar
  23'42
  $ echo $bar:q
  23´42
```

Convert the words to all lowercase.

```
  $ bar=FOOBAR
  $ echo $bar
  FOOBAR
  $ echo $bar:l
  foobar
```

Convert the words to all uppercase.

```
 $ bar=foobar
 $ echo $bar
 foobar
 $ echo $bar:u
 FOOBAR
```

Variables can be modifi ed by modifi ers, too. That makes modifi cation of variables possible without using any external program.

```
 sentence="beginning and some words of a sentence with end."
```

Now lets split this sentence-var by using the (s| |) modifi er which modifi es words by splitting at " ":

```
 words=${(s| |)sentence}
 print $words[1] -> "beginning"
 print $words[-1] ->"end."
```

Now if one wants to have the beginning of a sentence with a Capital, it's as easy as doing:

```
 print "${(C)words[1]} $words[2,-1]"
```

which capitalizes the fi rst word of the list words and then adds with " " second to last word of words. It's possible to join these words as a colon separated scalar.

```
 colonlist=${(j|,|)words} # (j|,|) joins with ",".
```

You can see that it's a scalar by testing with (t):

```
 print ${(t)colonlist} prints "scalar".
 print ${(t)words} prints "array".
```

It's possible to sort arrays with o and O:

```
 print ${(o)words} # lists the words-array sorted (forwards)
 print ${(O)words} # lists the words-array sorted (backwards)
```

COMPLETITION

See also *man 1 zshcompctl zshcompsys zshcompwid*. zshcompctl is the old style of zsh programmable completion, zshcompsys is the new completion system, zshcompwid are the zsh completion widgets.

Some functions, like _apt and _dpkg, are very slow. You can use a cache in order to proxy the list of results (like the list of available debian packages) Use a cache:

```
zstyle ':completion:*' use-cache on
zstyle ':completion:*' cache-path ~/.zsh/cache
```

Prevent CVS fi les/directories from being completed :

```
zstyle ':completion:*:(all-|)fi les' ignored-patterns '(|*/)CVS'
zstyle ':completion:*:cd:*' ignored-patterns '(*/)#CVS'
```

Fuzzy matching of completions for when you mistype them:

```
zstyle ':completion:*' completer _complete _match _approximate
zstyle ':completion:*:match:*' original only
zstyle ':completion:*:approximate:*' max-errors 1 numeric
```

And if you want the number of errors allowed by _approximate to increase with the length of what you have typed so far:

```
zstyle -e ':completion:*:approximate:*' max-errors 'reply=( $(( ($#PREFIX+$#SUFFIX)/3 )) numeric )'
```

Ignore completion functions for commands you don’t have:
```
zstyle ’:completion:*:functions’ ignored-patterns ’_*’
```

With helper functions like:
```
xdvi() { command xdvi ${*:-*.dvi(om[1])} }
```

you can avoid having to complete at all in many cases, but if you do, you might want to fall into menu selection immediately and to have the words sorted by time:
```
zstyle ’:completion:*:*:xdvi:*’ menu yes select
zstyle ’:completion:*:*:xdvi:*’ file-sort time
```

Completing process IDs with menu selection:
```
zstyle ’:completion:*:*:kill:*’ menu yes select
zstyle ’:completion:*:kill:*’   force-list always
```

If you end up using a directory as argument, this will remove the trailing slash (usefull in ln)
```
zstyle ’:completion:*’ squeeze-slashes true
```

cd will never select the parent directory (e.g.: cd ../<TAB>):
```
zstyle ’:completion:*:cd:*’ ignore-parents parent pwd
```

ADVANCED GLOBBING

See *man zshexpn | less -p ’Glob Qualifiers’*

List file ’foobar’ via recursiv search in directories:
```
$ ls **/foobar
```

List files file20, file30, file100, etc:
```
$ ls file<20->
```

List files with suffix c and pro (e.g. foo.c, bar.pro):
```
$ ls *.(c|pro)
```

List files which are word-readable:
```
$ ls *(R)
```

List all .c-files except ’lex.c’:
```
$ ls *.c˜lex.c
```

List all ‘README’ - files case-insensitive with max. one typo (e.g. RADME, REEME, RAEDME):
```
$ ls (#a1)README
```

List files named README but accept one spelling error including case-insensitive (e.g. RADME, REEME,

RAEDME):
```
 $ ls (#ia1)README
```

List executable files, directories and symlinks:
```
 $ ls *(*@)
```

List dangling symlinks:
```
 $ ls **/*(-@)
```

List all zero-length-files which are not group- or world-writable:
```
 $ ls *(L0f.go-w.)
```

List all .c-files for which there doesn’t exist a .o file:
```
 $ c=(*.c) o=(*.o(N)) eval 'ls ${${c:#(${~${(j:|:)${o:r}}}).c}:?done}'
```

Find (and print) all symbolic links without a target within the current dirtree:
```
 $ file **/*(D@) | fgrep broken
 $ for i in **/*(D@); [[ -f $i || -d $i ]] || echo $i
 $ echo **/*(@-^./=%p)
 $ print -l **/*(-@)
```

Rename all MP3-files from name with spaces.mp3 to Name With Spaces.mp3:
```
 $ for i in *.mp3; do
         mv $i ${${(C)i}:s/Mp3/mp3/}
   done
```

Rename all PDF-files from name.mp3 to Name.mp3 (lowercase to uppercase of first letter) without touching the rest of the filename:
```
 $ zmv '([a-z])(*).pdf' '${(C)1}$2.pdf'
```

Substitutions in strings can be done by string-indexes:
```
 $ a="doh.";a[1]='d';a[-1]='. (Bart Simpson)'
 $ echo $a
 doh. (Bart Simpson)
```

Associative arrays:
```
 $ typeset -A ass_array; ass_array=(one 1 two 2 three 3 four 4)
 $ print ${(k)ass_array} # prints keys
 one four two three
 $ print ${(v)ass_array} # prints values
 1 4 2 3
 $ print $ass_array[one]
 1
```

Extract parts of a string. Print ﬁ rst word of output of ’date’:
```
 $ print ${$( date )[1]}
```

Extract parts of a string. Print ip-address of loopback device:
```
 $ print ${${$( LC_ALL=C /sbin/ifconﬁ g lo )[6]}#addr:}
```

Print speciﬁ c line of a ﬁ le. E.g. print line 5 of ﬁ le:
```
 $ print -l ${"$( < ﬁ le )"[(f)5]}
```

Print line containing string ’root’ of ﬁ le /etc/passwd:
```
 $ print ${"$( < /etc/passwd )"[(fr)*root*]}
```

Print words two to four of output of ’date’:
```
 $ print ${$( date )[2,4]}
```

Use of two-dimensional indizes. Print time via date but without seconds:
```
 $ print ${$(date)[4][1,5]}
```

Calculate ﬂoating point numbers:
```
 $ printf "%.0f0 $[ 2.8*15 ]
```

Convert images from foo.gif to foo.png:
```
 $ for i in **/*.gif; convert $i $i:r.png
```

Download ﬁ les created with LaTeX2HTML (e.g. the ZSH-Guide):
```
 $ for f in http://zsh.sunsite.dk/Guide/zshguide{,{01..08}}.html; do
     lynx -source $f >${f:t}
   done
```

Make with dpkg a master-list of everyﬁ le that it has installed:
```
 $ diff <(ﬁ nd / | sort) <(cat /var/lib/dpkg/info/*.list | sort)
```

Replace this color escape-sequences:
```
 $ autoload colors ; colors
 $ print "$bg[cyan]$fg[blue]Welcome to man zsh-lovers" >> $TTY
```

Get ASCII value of a character:
```
 $ char=N ; print $((#char))
```

Filename suffix. Note: (N) activates setopt nullglob only for this loop.
```
 $ for i in *.o(N); do
         rm $i
   done
```

Rename files: 'FOO' to 'foo':
```
 $ for i in *(.); mv $i ${i:l}
```

Rename files: 'bar' to 'BAR':
```
 $ for i in *(.); mv $i ${i:u}
```

Show all suid-files in $PATH:
```
 $ ls -latg ${(s.:.)PATH} | grep '^...s'
```

ZMV - multiple move with zsh

Requires 'autoload zmv'. Some examples:

Move serially all files (foo.foo > 1.foo, fnord.foo > 2.foo, ..).
```
 $ ls *
 1.c  asd.foo  bla.foo  fnord.foo  foo.fnord  foo.foo
 $ c=1 zmv '*.foo' '$((c++)).foo'
 $ ls *
 1.c  1.foo  2.foo  3.foo  4.foo  foo.fnord
```

See above, but now only files with a filename >= 30 chars.
```
 $ c=1 zmv "${(l:30-4::?:)}*.foo" '$((c++)).foo'
```

Replace spaces in filenames with a underline.
```
 $ zmv '* *' '$f:gs/ /_'
```

Change the suffix from *.sh to *.pl.
```
 $ zmv -W '*.sh' '*.pl'
```

Lowercase/uppercase all files and directories.
```
 $ zmv '(*)' '${(L)1}' for lowercase
 $ zmv '(*)' '${(U)1}' for uppercase
```

Remove the suffix *.c from all c-files.
```
 $ zmv '(*).c' '$1'
```

Uppercase only the first letter of all *.mp3 - files.
```
 $ zmv '([a-z])(*).mp3' '${(C)1}$2.mp3'
```

Copy the target ‘README’ in same directory as each ‘Makefile’.

```
 $ zmv -C ’(**/)Makefile’ ’${1}README’
```

Rename pic1.jpg, pic2.jpg,.. to pic0001.jpg, pic0002.jpg,...

```
 $ zmv ’pic(*).jpg’ ’pic${(l:4::0:)1}.jpg’
 $ zmv ’(**/)pic(*).jpg’ ’$1/pic${(l:4::0:)2}.jpg’ # recursive
```

MODULES

See also *man zshmodules*. Don’t forget to run *zmodload −i MODULENAME* before using a module. Example: *zmodload -i zsh/datetime*.

zsh/cap

Builtins for manipulating POSIX.1e (POSIX.6) capability (privilege) sets.

zsh/clone

A builtin that can clone a running shell onto another terminal.

Creates a forked instance of the current shell ($! is set to zero) and execute ‘‘command’’ on /dev/tty8 (for this example):

```
 $ zmodload zsh/clone
 $ clone /dev/tty8 && (($! == 0)) && exec command
```

zsh/compctl

The **compctl** builtin for controlling completion.

zsh/complete

The basic completion code.

zsh/complist

Completion listing extensions.

zsh/computil

A module with utility builtins needed for the shell function based completion system.

zsh/datetime

Some date/time commands and parameters.

Do not have GNU date? Let’s replace it:

```
 $ alias datereplacement=’strftime "%Y-%m-%d" $EPOCHSECONDS’
 $ export DATE=‘datereplacement‘
 $ echo $DATE
```

zsh/deltochar

A ZLE function duplicating EMACS’ **zap−to−char**.

zsh/example

An example of how to write a module.

zsh/files

Some basic file manipulation commands as builtins.

```
# search a directory for files containing a certain string then copy those files to another directory.
  $ IFS=$’  ’
  $ cp $(grep -lZr foobar .) otherdirectory
```

zsh/mapfile

Access to external files via a special associative array.

```
# grepping for two patterns
  $ pattern1="foo"
  $ pattern2="bar foo"
  $ print -l ./**/*(DN.e{'z=$mapfile[$REPLY] &&
  [[ $z = *$pattern1* && $z = *$pattern2* ]]'})
# or a solution in combination with zsh/pcre
  $ zmodload -i zsh/mapfile zsh/pcre
  $ pattern1="foo"
  $ pattern2="bar foo"
  $ pcre_compile "(?s)(?=.*?$pattern1).*?$pattern2"
  $ pcre_study
  $ print -l ./**/*(DN.e{'pcre_match $mapfile[$REPLY]'})

# equivalent for ``less /etc/passwd | grep -v root''
  $ IFS=$'0
  $ print -rl -- ${${=mapfile[/etc/passwd]}:#*root*}
# or - for case insensitive
  $ setopt extendedglob
  $ print -rl -- ${${=mapfile[/etc/passwd]}:#*(#i)root*}

# If a XML-file contains stuff like ``<TAGA/>'' and ``<TAGB/>'', number this empty tags
# (ones ending in '/>') so if encountered in the same order, the preceeding tags would become
# ``<TAGA/>1</TAGA>'' and ``<TAGB/>2</TAGB>''
  $ cnt=0
  $ apfile[data.xml.new]=${(S)mapfile[data.xml]//  > (#im)<TAGA>*<TAGA>/<TAGA>$((++cnt))<TAGA>}

# removing all files in users Maildir/new that contain ``filename="gone.src''
  $ zmodload zsh/{files,mapfile}
  $ rm -f /u1/??/*/Maildir/new/100*(.e{'[[ $mapfile[$REPLY] == *filename=

# Grep out the Title from a postscript file and append that value to the end of
# the filename
  $ autoload -U zmv
  $ zmv '(*).ps' '$1-${${${mapfile[$f]##*%%Title: }%% *}//[^a-zA-Z0-9_]/}.ps'
```

zsh/mathfunc

Standard scientific functions for use in mathematical evaluations.

```
$ echo $(( sin(1/4.0)**2 + cos(1/4.0)**2 - 1 ))
  -1.1102230246251565e-16
$ echo $(( pi = 4.0 * atan(1.0) ))
  3.1415926535897931
$ echo $(( f = sin(0.3) ))
  0.29552020666133955
$ print $(( rand48(seed) ))
  0.01043488334700271
```

zsh/parameter

Access to internal hash tables via special associative arrays.

zsh/pcre

Interface to the PCRE library.

Important: requires zsh compiled with pcre-support. Check whether your version supports pcre via 'ldd =zsh | grep pcre'. PCRE provides support for Perl's regular expressions (regex). You have to compile a regex and can match it afterwards using error codes:

```
 $ zmodload zsh/pcre
 $ pcre_compile '\s\d.\d{3}.\d{3} Euro'  &&\
   pcre_match ' 1.000.000 Euro' &&\
   echo "matches" || echo "does not match"
```

Note: if you are using complex regular expressions you can improve speed via pcre_study.

zsh/sched

A builtin that provides a timed execution facility within the shell.

zsh/net/socket

Manipulation of Unix domain sockets

```
 $ zmodload zsh/net/socket
 $ zsocket -l -d 3
 # ``-l'': open a socket listening on filename
 # ``-d'': argument will be taken as the target file descriptor for the
 #         connection
 # ``3'' : file descriptor. See ``A User's Guide to the Z-Shell''
 #         (3.7.2: File descriptors)
 $ zsocket -a -d 4 3
 # ``-a'': accept an incoming connection to the socket
 $ zsocket -a -d 5 3 # accept a connection
 $ echo foobar >&4
 $ echo barfoo >&5
 $ 4>&- 5>&- 3>&-
```

In one shell:

```
 $ zmodload zsh/net/socket
 $ zsocket -l -d 3 /tmp/mysocket # open listening socket
 $ zsocket -a -d 4 3             # accept a connection
 $ zsocket -a -d 5 3             # accept a connection
 $ echo Hi there >&4
 $ echo Hi there also >&5
 $ exec 4>&- 5>&- 3>&-
```

In another shell:

```
 $ zmodload zsh/net/socket
 $ zsocket -d 3 /tmp/mysocket # connect to /tmp/socket
 $ zsocket -d 4 /tmp/mysocket # connect to /tmp/socket
 $ read msg <&3; echo got: "$msg on fd 3"
 $ read msg <&4; echo got: "$msg on fd 4"
 $ exec 3>&- 4>&-
```

zsh/stat

A builtin command interface to the **stat** system call.

Get size of a file in bytes:

```
 $ zmodload -i zsh/stat
 $ stat -L +size file
```

Equal to GNU's:

```
 $ stat -c %s file
```

Comparing file dates:

```
  $ file1=foo
  $ file2=bar
  $ touch bar & sleep 5 & touch foo
  $ echo $file1 is $(( $(stat +mtime $file2) - $(stat +mtime $file1) )) seconds older than $file2.
  bar is 5 seconds older than foo
```

List the files of a disk smaller than some other file:

```
$ stat -A max +size some-other-file
$ print -rl ./**/*(D.L-$max)
```

List the top 100 biggest files in a disk:

```
$ ls -fld ./**/*(d‘stat +device .‘OL[1,100])
```

Get only the user name and the file names from (like ls -l * | awk '{print $3" " $8}'):

```
$ for file; do
>   stat -sA user +uid -- "$file" &&
>     print -r -- "$user" "$file"
> done
```

Get the difference between actual bytes of file and allocated bytes of file:

```
$ print $(($(stat +block -- file) * 512 - $(stat +size -- file)))
```

Find largest file:

```
$ stat +size ./*(DOL[1])
# ‘‘D’’  : to include dot files (d lowercase is for device)
# ‘‘O’’  : reverse Ordered (o lowercase for non-reverse order)
# ‘‘L’’  : by file Length (l is for number of links)
# ‘‘[1]’’: return only first one
```

Delete files in a directory that hasn't been accessed in the last ten days and send ONE mail to the owner of the files informing him/her of the files' deletion:

```
$ zmodload zsh/stat zsh/files
$ typeset -A f; f=()
$ rm -f /path/**/*(.a+10e{'stat -sA u +uidr $REPLY; f[$u]="$f[$u]$REPLY"'})
$ for user (${(k)f}) {print -rn $f[$user]|mailx -s "..." $user}
```

Get a "ls -l" on all the files in the tree that are younger than a specified age:

```
$ for d (. ./**/*(N/m-2))
>   print -r -- $'0$d: && cd $d && {
>      for f (*(Nm-2om))
>   stat -F '%b %d %H:%M' -LsAs -- $f &&
>   print -r -- $s[3] ${(l:4:)s[4]} ${(l:8:)s[5]} \
>   ${(l:8:)s[6]} ${(l:8:)s[8]} $s[10] $f ${s[14]:+-> $s[14]}
>   cd ~-
> }
```

Get file creation date:

```
$ stat -F '%d %m %Y' +mtime ~/.zshrc
30 06 2004
$ stat -F '%D' +mtime ~/.zshrc
06/30/04
```

zsh/system
A builtin interface to various low−level system features.

zsh/net/tcp
Manipulation of TCP sockets

zsh/termcap
Interface to the termcap database.

```
$ zmodload -ab zsh/termcap echotc
$ GREEN=‘echotc AF 2‘
$ YELLOW=‘echotc AF 3‘
```

```
 $ RED=‘echotc AF 1‘
 $ BRIGHTRED=‘echotc md ; echotc AF 1‘
 $ print -l ${GREEN}green ${YELLOW}yellow ${RED}red ${BRIGHTRED}brightred
```

zsh/terminfo
Interface to the terminfo database.

zsh/zftp
A builtin FTP client.

Write ftp scripts as though shell:

```
 $ init
 $ autoload -U zfinit && zfinit
 $ zfparams www.example.invalid myuserid mypassword
 $ zfopen
 $ zfcd tips
 $ zfls -l zsh-lovers.html
 $ zfput zsh-lovers.html
 $ zfls -l zsh-lovers.html
```

Automatically transfer files using FTP with error checking:

```
  $ zftp open host.name.invalid user passwd || exit
  $ zftp get /remote/file > /local/file; r=$?
  $ zftp close && exit r
```

Compress and ftp on the fly:

```
  $ zftp open host.name.invalid user password
  $ zftp get $file | bzip2 > ${file}.bz2
  $ zftp close
```

Long list of files on a ftp:

```
  $ autoload -U zfinit
  $ zfinit
  $ zfopen some-host
  $ zfcd /some/remote/Dir
  $ cd /some/local/Dir
```

If the list.txt is located on the remote host, change to

```
  $ zfget ${(f)"$(zftp get /path/to/remote/list.txt)"}
  $ zfget ${(f)"$(cat list.txt)"}
  $ zfclose
```

zsh/zle The Zsh Line Editor, including the **bindkey** and **vared** builtins.

zsh/zleparameter
Access to internals of the Zsh Line Editor via parameters.

zsh/zprof
A module allowing profiling for shell functions.

zsh/zpty
A builtin for starting a command in a pseudo−terminal.

```
 $ zmodload -i zsh/zpty
 $ zpty PW passwd $1
 # “-r”: read the output of the command name.
 # “z” : Parameter
 $ zpty -r PW z ’*password:’
 # send the to command name the given strings as input
```

```
$ zpty -w PW $2
$ zpty -r PW z '*password:'
$ zpty -w PW $2
# | The second form, with the -d option, is used to delete commands
# | previously started, by supplying a list of their names. If no names
# | are given, all commands are deleted. Deleting a command causes the HUP
# | signal to be sent to the corresponding process.
$ zpty -d PW
```

zsh/zselect

Block and return when file descriptors are ready.

```
# It's simular to
 ,----
 | $ sg=$(stty -g)
 | $ stty -icanon min 0 time 50
 | $ read yesno
 | $ stty "$sg"
 | $ case "$yesno" in
 | >  yes) command1;;
 | >  *) command2;;
 | > esac
 `----
$ if zselect -t 500 -r 0 && read yesno && [ yes = "$yesno" ]; then
>    command1
> else
>    command1
> fi
```

zsh/zutil

Some utility builtins, e.g. the one for supporting configuration via styles.

SUBSTITUTION

Path substitution:

```
$ ls -l =zsh  # is like: 'ls -l /path/to/zsh' or 'ls -l `which zsh`'
```

Process substitution:

```
$ (vi =(cmd)) # edit output of 'cmd' (called process substitution).
```

Substitution of variables:

```
$ var1=42
$ tmp=var1
$ echo $((tmp))
42
$

$ var=foo
$ tmp=var
$ echo ${(P)tmp}
foo
```

ALIASES

Suffix aliases are supported in zsh since version 4.2.0. Some examples:

```
alias -s tex=vim
alias -s html=w3m
alias -s org=w3m
```

Now pressing return-key after entering 'foobar.vim' starts vim with foobar.vim. Calling a html-file runs browser w3m. 'www.zsh.org' and pressing enter starts w3m with argument www.zsh.org.

Global aliases can be used anywhere in the command line. Example:

```
$ alias -g C='| wc -l'
$ grep alias ~ /.zsh/* C
443
```

Some more or less useful global aliases (choose whether they are useful or not for you on your own):

```
alias -g ...='../..'
alias -g ....='../../..'
alias -g .....='../../../..'
alias -g CA="2>&1 | cat -A"
alias -g C='| wc -l'
alias -g D="DISPLAY=:0.0"
alias -g DN=/dev/null
alias -g ED="export DISPLAY=:0.0"
alias -g EG='|& egrep'
alias -g EH='|& head'
alias -g EL='|& less'
alias -g ELS='|& less -S'
alias -g ETL='|& tail -20'
alias -g ET='|& tail'
alias -g F=' | fmt -'
alias -g G='| egrep'
alias -g H='| head'
alias -g HL='|& head -20'
alias -g §k="*~ (*.bz2|*.gz|*.tgz|*.zip|*.z)"
alias -g LL="2>&1 | less"
alias -g L="| less"
alias -g LS='| less -S'
alias -g MM='| most'
alias -g M='| more'
alias -g NE="2> /dev/null"
alias -g NS='| sort -n'
alias -g NUL="> /dev/null 2>&1"
alias -g PIPE='|'
alias -g R=' > /c/aaa/tee.txt '
alias -g RNS='| sort -nr'
alias -g S='| sort'
alias -g TL='| tail -20'
alias -g T='| tail'
alias -g US='| sort -u'
alias -g VM=/var/log/messages
alias -g X0G='| xargs -0 egrep'
alias -g X0='| xargs -0'
alias -g XG='| xargs egrep'
alias -g X='| xargs'
```

Array parameters [array_name=(value1 value2 ... valueN)].

```
$ stupid=emacs
$ echo $stupid[3]
```

```
 a
 $
```

SHELL-SCRIPTING

This section provides some examples for often needed shellscript-stuff. Notice that you should not use otherwise most examples won’t work.

Parse options in shellscripts. Example taken from ZWS by Adam Chodorowski (http://www.chodorowski.com/projects/zws/):

```
parse_options()
{
   o_port=(-p 9999)
   o_root=(-r WWW)
   o_log=(-d ZWS.log)

   zparseopts -K -- p:=o_port r:=o_root h=o_help
   if [[ $? != 0 || "$o_help" != "" ]]; then
      echo Usage: $(basename "$0") "[-p PORT] [-r DIRECTORY]"
      exit 1
   fi

   port=$o_port[2]
   root=$o_root[2]
   log=$o_log[2]

   if [[ $root[1] != '/' ]]; then root="$PWD/$root"; fi
}
# now use the function:
parse_options $*
```

MISC−EXAMPLES

Hint: A list of valid glob Qualifiers can be found in zshexpn(1). See ‘‘man 1 zshexpn | less -p’’ Qualifiers for details.

Load all available modules at startup

```
 $ typeset -U m
 $ m=()
 $ for md ($module_path) m=($m $md/**/*(*e:'REPLY=${REPLY#$md/}'::r))
 $ zmodload -i $m
```

Rename all MP3-Files from ‘‘name with spaces.mp3’’ to ‘‘Name With Spaces.mp3’’:

```
 $ for i in *.mp3; do
 >     mv $i ${${(C)i}:s/Mp3/mp3/}
 > done
```

Download with LaTeX2HTML created Files (for example the ZSH−Guide):

```
 $ for f in http://zsh.sunsite.dk/Guide/zshguide{,{01..08}}.html; do
 >     lynx -source $f >${f:t}
 > done
```

Replace the unreadable Escape-Sequences:

```
 $ autoload colors ; colors
 $ print "$bg[cyan]$fg[blue]You are an zsh user" >> /dev/pts/3
```

Filename−Expansion. **Note:** (N) activates setopt nullglob only for this loop.

```
 $ for i in *.o(N); do
 >     rm $i
 > done
```

Re-linking broken links:

```
 $ for f in ./**/*(-@); do
 >     stat +link -A l $f
 >     (cd $f:h & [[ -e $l.gz ]]) & ln -sf $l.gz $f
 > done
```

Show me all the .c files for which there doesn’t exist a .o file:

```
  $ c=(*.c) o=(*.o(N)) eval ’ls ${${c:#(${˜ ${(j:|:)${o:r}}}).c}:?done}’
```

Load all available modules at startup:

```
  $ typeset -U m
  $ m=()
  $ for md ($module_path) m=($m $md/**/*(*e:’REPLY=${REPLY#$md/}’::r))
  $ zmodload -i $m
```

Rename all files within a directory such that their names get a numeral prefix in the default sort order:

```
  $ i=1; for j in *; do mv $j $i.$j; ((i++)); done
  $ i=1; for f in *; do mv $f $(echo $i| awk ’{ printf("%03d", $0)}’).$f; ((i++)); done
  $ integer i=0; for f in *; do mv $f $[i+=1].$f; done
```

Find (and print) all symbolic links without a target within the current dirtree:

```
  $ $ file **/*(D@) | fgrep broken
  $ for i in **/*(D@); [[ -f $i || -d $i ]] || echo $i
  $ echo **/*(@-ˆ./=%p)
  $ print -l **/*(-@)
```

List all plain files that do not have extensions listed in ‘fignore’:

```
  $ ls **/*˜ *(${˜ ${(j/|/)fignore}})(.)
  # see above, but now omit executables
  $ ls **/*˜ *(${˜ ${(j/|/)fignore}})(.ˆ*)
```

Print out files that dont have extensions (require setopt extendedglob dotglob):

```
  $ printf ’%s0 ˆ?*.*
```

List files in reverse order sorted by name:

```
  $ print -rl -- *(On)
  or
  $ print -rl -- *(ˆon)
```

Synonymic to ‘‘ps ax | awk ’{print $1}’’:

```
  $ print -l /proc/*/cwd(:h:t:s/self//)
```

Get the PID of a process (without ‘‘ps’’, ‘‘sed’’, ‘‘pgrep’’, .. (under Linux):

```
  $ pid2 () {
  >   local i
  >   for i in /proc/<->/stat
  > do
  >   [[ "$(< $i)" = *\((${(j:|:)˜ @})\)* ]] && echo $i:h:t
  > done
  > }
```

for X in ’n’ ’o’ ’p’ ’q’ ’r’ ’s’ ’t’ ’u’ ’v’ ’w’ ’x’ ’y’; do ...:

```
  $ for (( i = 36#n; i <= 36#y; i++ )); do
  >   print ${$(([##36]i)):l}
  > done
# or in combination with ‘‘dc’’
  $ print {$((##n))..$((##y))}P 10P | dc
# or with ‘‘eval’’
  $ eval print ’${$(([##36]’{$((36#n))..$((36#y))}’)):l}’
```

Foreach in one line of shell:

```
  $ for f (*) print -r -- $f
```

Copy a directory recursively without data/files:

```
  $ dirs=(**/*(/))
  $ cd -- $dest_root
  $ mkdir -p -- $dirs
# or without zsh
  $ find . -type d -exec env d="$dest_root" sh -c ’ exec mkdir -p -- "$d/$1"’ ’{}’ ’{}’ ;
```

If ‘foo=23’’, then print with 10 digit with leading ’0’:

```
  $ foo=23
  $ print ${(r:10::0:)foo}
```

Find the name of all the files in their home directory that have more than 20 characters in their file names:

```
  print -rl $HOME/${(l:20::?:)~ :-}*
```

Save arrays:

```
  $ print -r -- ${(qq)m} > $nameoffile      # save it
  $ eval "m=($(cat -- $nameoffile)"            # or use
  $ m=("${(@Q)${(z)"$(cat -- $nameoffile)"}}") # to restore it
```

Get a "ls -l" on all the files in the tree that are younger than a specified age (e.g "ls -l" all the files in the tree that where modified in the last 2 days):

```
  $ ls -tld **/*(m-2)
```

This will give you a listing 1 file perl line (not à la ls -R). Think of an easy way to have a "ls -R" style output with only files newer than 2 day old.

```
  $ for d (. ./**/*(/)) {
  >   print -r -- $’0${d}:
  >   cd $d && {
  >       l=(*(Nm-2))
  >       (($#l)) && ls -ltd -- $l
  >       cd ~ -
  >   }
  > }
```

If you also want directories to be included even if their mtime is more than 2 days old:

```
  $ for d (. ./**/*(/)) {
  >   print -r -- $’0${d}:
  >   cd $d && {
  >      l=(*(N/,m-2))
  >      (($#l)) && ls -ltd -- $l
```

```
 >     cd ˜ -
 >   }
 > }
```

And if you want only the directories with mtime < 2 days to be listed:

```
 $ for d (. ./**/*(N/m-2)) {
 >   print -r -- $'0${d}:
 >   cd $d && {
 >     l=(*(Nm-2))
 >     (($#l)) && ls -ltd -- $l
 >     cd ˜ -
 >   }
 > }
```

Print 42 “-”:

```
 $ echo ${(l:42::-:)}
# or use “$COLUMS”
 $ echo ${(l:$COLUMNS::-:)}
# and now with colors (require autoload colors ;colors)
 $ echo "$bg[red]$fg[black]${(l:42::-:)}"
```

Redirect STDERR to a command like xless without redirecting STDOUT as well:

```
 $ foo 2>>(xless)
# but this executes the command asynchronously. To do it synchronously:
 $ { { foo 1>&3 } 2>&1 | xless } 3>&1
```

Rename all MP3-Files from name with spaces.mp3 to Name With Spaces.mp3:

```
 $ for i in *.mp3; do
 >    mv $i ${${(C)i}:s/Mp3/mp3/}
 > done
```

Match file names containing only digits and ending with .xml (requiresetopt kshglob):

```
 $ ls -l [0-9]##.xml
 $ ls -l <0->.xml
```

Remove all "non txt" files:

```
 $ rm ./ˆ *.txt
```

Move 200 files from a directory into another:

```
 $ mv -- *([1,200]) /another/Dir
```

Convert images (foo.gif => foo.png):

```
 $ for i in **/*.gif; convert $i $i:r.png
```

Convert a collection of mp3 files to wave or cdr (e.g. file.wav -> file.mp3):

```
 $ for i (./*.mp3){mpg321 --w - $i > ${i:r}.wav}
```

Download with LaTeX2HTML created Files (for example the ZSH-Guide):

```
 $ for f in http://zsh.sunsite.dk/Guide/zshguide{,{01..08}}.html; do
 >    lynx -source $f >${f:t}
 > done
```

Move all files in dir1 and dir2 that have line counts greater than 10 to another directory say "/more10":

```
 $ mv dir[12]/**/*.cr(-.e{'((‘wc -l < $REPLY‘ > 10))'}) /more10
```

Make with dpkg a master-list of everyfi le that it has installed:
```
 $ diff <(fi nd / | sort) <(cat /var/lib/dpkg/info/*.list | sort)
```

Replace the unreadable Escape-Sequences:
```
 $ autoload colors ; colors
 $ print "$bg[cyan]$fg[blue]You are an zsh user" >> /dev/pts/3
```

Get ASCII value of a character:
```
 $ char=N ; print $((#char))
```

Filename suffi x: Note: The (N) says to use the nullglob option for this particular glob pattern.
```
 $ for i in *.o(N); do
 >     rm $i
 > done
```

Rename fi les; i. e. FOO to foo and bar to BAR:
```
 $ for i in *(.); mv $i ${i:l} # ‘FOO’ to ‘foo’
 $ for i in *(.); mv $i ${i:u} # ‘bar to ‘BAR’
```

Show all suid-fi les in $PATH:
```
 $ ls -latg ${(s.:.)PATH} | grep ’ˆ ...s’
# or more complex ;)
 $ print -l ${ˆ path}/*(Ns,S)
# or show only executables with a user given pattern
 $ print -l ${ˆ path}/*vim*(*N)
```

gzip fi les when containing a certain string:
```
 $ gzip ${(ps: :)"$(grep -lZ foobar ./*.txt(.))"}
```

A small one-liner, that reads from stdin and prints to stdout the fi rst unique line i. e. does not print lines that have been printed before (this is similar to the unique command, but unique can only handle adjacent lines):
```
 $ IFS=$’0; print -rl -- ${(Oau)${(Oa)$(cat fi le;echo .)[1,-2]}}
```

Lists every executable in PATH:
```
 $ print -l ${ˆ path}/*(-*N)
```

Match all .c fi les in all subdirectories, _except_ any SCCS subdirectories?
```
 $ ls **/*.c˜ (*/)#SCCS/*
```

List all ‘README’ - fi les case-insensitive with max. one typo:
```
 $ ls **/*(#ia2)readme
```

Print version information of zsh:
```
 $ print $ZSH_VERSION
```

Get hostspecifi c information:
```
 $ echo $MACHTYPE $VENDOR $OSTYPE
```

Fast change of directories:
```
 alias ...=’cd ../..’
 alias ....=’cd ../../..’
 alias .....='cd ../../../..'
 alias ......='cd ../../../../..'
 alias .......='cd ../../../../../..'
```

Mailpath: simple multiple mailpath:

```
  mailpath=($HOME/Mail/mbox'?new mail in mbox'
            $HOME/Mail/tux.u-strasbg'?new mail in tux'
            $HOME/Mail/lilo'?new mail in lilo'
            $HOME/Mail/ldap-fr'?new mail in ldap-fr')
```

Mailpath: dynamic mailpath:

```
  typeset -a mailpath
  for i in ˜ /Mail/Lists/*(.); do
     mailpath[$#mailpath+1]="${i}?You have new mail in ${i:t}."
  done
```

Avoid globbing on special commands:

```
for com in alias expr find mattrib mcopy mdir mdel which;
alias $com="noglob $com"
```

For migrating your bashprompt to zsh use the script bash2zshprompt located in the zsh source distribution under 'Misc'.

For migration from (t)csh to zsh use the c2z tool that converts csh aliases and environment and shell variables to zsh. It does this by running csh, and having csh report on aliases and variables. The script then converts these to zsh startup files. It has some issues and usage information that are documented at the top of this script.

Here are functions to set the title and hardstatus of an **XTerm** or of **GNU Screen** to 'zsh' and the current directory, respectively, when the prompt is displayed, and to the command name and rest of the command line, respectively, when a command is executed:

```
  function title {
     if [[ $TERM == "screen" ]]; then
       # Use these two for GNU Screen:
       print -nR $'  33k'$1$'  33'\
       print -nR $'  33]0;'$2$''
     elif [[ $TERM == "xterm" || $TERM == "rxvt" ]]; then
       # Use this one instead for XTerms:
       print -nR $'  33]0;'$*$''
     fi
  }

  function precmd {
     title zsh "$PWD"
  }

  function preexec {
     emulate -L zsh
```

```
      local -a cmd; cmd=(${(z)1})
      title $cmd[1]:t "$cmd[2,-1]"
  }
```

Put the following line into your ˜ /.screenrc to see this fancy hardstatus:

```
  caption always "%3n %t%? (%u)%?%?: %h%?"
```

Special variables which are assigned or you can assign:

```
 $ echo $LINENO $RANDOM $SECONDS $COLUMNS $HISTCHARS
 $ echo $UID $EUID $GID $EGID $USERNAME
 $ echo $fignore $mailpath $cdpath
```

Show me all the .c files for which there doesn’t exist a .o file:

```
 $ c=(*.c) o=(*.o(N)) eval ’ls ${${c:#(${˜ ${(j:|:)${o:r}}}).c}:?done}’
```

Find (and print) all symbolic links without a target within the current dirtree:

```
 $ file **/*(D@) | fgrep broken
 $ for i in **/*(D@); [[ −f $i || −d $i ]] || echo $i
 $ echo **/*(@−ˆ ./=%p)
 $ print −l **/*(−@)
```

Rename files; i. e. FOO to foo and bar to BAR:

```
 $ for i in *(.); mv $i ${i:l} # ‘FOO’ to ‘foo’
 $ for i in *(.); mv $i ${i:u} # ‘bar to ‘BAR’
```

Show all suid-files in $PATH:

```
 $ ls −latg ${(s.:.)PATH} | grep ’ˆ ...s’
```

List all ‘README’ - files case-insensitive with max. one typo:

```
 $ ls **/*(#ia2)readme
```

(RECURSIVE) GLOBBING−EXAMPLES

Search for ‘README’ in all Subdirectories

```
 $ print −l **/README
```

Recursive ‘‘chmod’’

```
 $ chmod 700 **/(.) # Only files
 $ chmod 700 **/(/) # Only directories
```

List files beginning at ‘foo23’ upwards (foo23, foo24, foo25, ..)

```
 $ ls -l foo<23−>
```

Remove spaces from filenames

```
 $ for a in ./**/*\ *(Dod); do mv $a ${a:h}/${a:t:gs/ /_}; done
```

Show only all *.c and *.h - Files

```
 $ ls -l *.(c|h)
```

Show **only** all *.c - files and ignore ‘foo.c’

```
 $ ls *.c˜ foo.c
```

```
Show only world-readable files
 $ ls -l *(R)

find and delete the files which are older than a given parameter (seconds/minutes/hours)
 # deletes all regular file in /Dir that are older than 3 hours
   $ rm -f /Dir/**/*(.mh+3)
 # # deletes all symlinks in /Dir that are older than 3 minutes
   $ rm -f /Dir/**/*(@mm+3)
 # deletes all non dirs in /Dir that are older than 30 seconds
   $ rm -f /Dir/**/*(ms+30^ /)
 # deletes all files more than 6 hours old
   $ rm -f **/*(mh+6)
 # deletes all folders, sub-folders and files older than one hour
   $ rm ./**/*(.Dmh+1,.DL0)
 # removes all files but the ten newer ones (delete all but last 10 files in a directory)
  $ rm ./*(Om[1,-11])

Note: If you get a arg list too long, you use the builtin rm. For example:
  $ zmodload zsh/files ; rm -f **/*(mh+6)
  or use the zargs function:
  $ autoload zargs ; zargs **/*(mh+6) -- rm -f
```

Explanation:

```
   ./: to avoid problem with files starting with "−"
  **/: recursively descend
   *.: any file
(...): qualifiers:
                (<a>,<b>): files of <a> type or <b> type
                 <a>:
                     .: regular files
                     D: including dot files
                  mh+1: whose [m]odification time, is more (+) than [1]
                        [h]our in the past.
                 <b>:
                    .: regular files
                    D: including dot files
                    L0: of 0 [L]ength.

If you want to remove empty directories afterwards:
 # ‘‘/’’ matches only directories and ‘‘od’’ sorted in depth order (so
 # that dir/subdir is removed before directory).
 $ rmdir ./**/*(/od) 2> /dev/null
```

Note: If you get a arg list too long, you use the builtin rm. For example:

```
 $ zmodload zsh/files ; rm -f **/*(mh+6)
or use the zargs function:
 $ autoload zargs ; zargs **/*(mh+6) -- rm -f

Delete only the oldest file in a directory:
 $ rm ./*filename*(Om[1])

Sort the output from ‘ls −l’ by file size:
 $ ls -fld *(OL)
```

Find most recent fi le in a directory:
```
 $ setopt dotglob ; print directory/**/*(om[1])
```

List the top 100 biggest fi les in a disk
```
 $ zmodload −i zsh/stat ; ls −fld ./**/*(d‘stat +device .‘OL[1,100])
 $ ls *(L0f.go-w.)
```

Find all fi les without a valid owner:
```
 $ chmod someuser /**/*(D^ u:${(j.:u:.)${(f)"$(</etc/passwd)"}%%:*}:)
```

Show only fi les are owned from group ‘users’:
```
 $ ls -l *(G[users])
```

ZMV−EXAMPLES

Note: “autoload zmv” needed! See “man zshcontrib | less -p zmv” for more details.

Serially all fi les (foo.foo > 1.foo, fnord.foo > 2.foo, ..):
```
 $ ls *
 1.c  asd.foo  bla.foo  fnord.foo  foo.fnord  foo.foo
 $ c=1 zmv ’*.foo’ ’$((c++)).foo’
 $ ls *
 1.c  1.foo  2.foo  3.foo  4.foo  foo.fnord
```

See above, but now only fi les with a fi lename >= 30 chars:
```
 $ c=1 zmv "${(l:30-4::?:)}*.foo" ’$((c++)).foo’
```

Replace spaces in fi lenames with a underline:
```
 $ zmv ’* *’ ’$f:gs/ /_’
```

Change the suffi x from *.sh to *.pl:
```
 $ zmv -W ’*.sh’ ’*.pl’
```

lowercase/uppercase all fi les/directories:
```
 # lowercase
   $ zmv ’(*)’ ’${(L)1}’
 # uppercase
   zmv ’(*)’ ’${(U)1}’
```

Remove the suffi x *.c from all C-Files:
```
 $ zmv ’(*).c’ ’$1’
```

Uppercase only the fi rst letter of all *.mp3 - fi les:
```
 $ zmv ’([a-z])(*).mp3’ ’${(C)1}$2.mp3’
```

Copy the target ‘README’ in same directory as each ‘Makefi le’:
```
 $ zmv -C ’(**/)Makefi le’ ’${1}README’
```

Removing single quote from fi lenames (recursive):
```
$ zmv -Q "(**/)(*’*)(D)" "\$1\${2//’/}"
```

Replace spaces with underscores in fi lenames (recursive):
```
$ zmv -Q "(**/)(* *)(D)" "\$1\${2// /_}"
```

Rename pic1.jpg, pic2.jpg, .. to pic0001.jpg, pic0002.jpg, ..:

```
# Not recursively
  $ zmv 'pic(*).jpg' 'pic${(l:4::0:)1}.jpg'
# Recursively
  $ zmv '(**/)pic(*).jpg' '$1/pic${(l:4::0:)2}.jpg'
```

TIPS BY ZZAPPER (http://www.rayninfo.co.uk/tips/zshtips.html)

```
!! #  last command
!$ #  last argument
!$:h (last argument, strip one level)
!?echo
vi !* (all parameters)
vi !$ (last parameters)
!42
history
^fred^joe             # edit previous command replace fred by joe
!42:p
also use control-R

cmdy !?cmd1?:*<TAB>   #get parameters of a previous command


!:0 is the previous command name
!^, !:2, !:3, ?, !$ are the arguments
!* is all the arguments
!-2, !-3, ? are earlier commands
!-2^, !-2:2, !-2$, !-2*

cd !$:h  (remove file name)
cat !!:t (only file name)
print ${param:&}   (last substitute)


# globbing modifiers
# :r removes the suffix from the result,
# :t takes away the directory part
# . means must be regular files not directories etc
# *(om[1]) picks most recently modified file
# (.N) no warning message if any file absent
print *(om[1])   # print the most recent file
print *(.om[1])  # print the most recent file (not directory)
ls -l *(Om[1])   # oldest file
print *(om[1,5]) # print the 5 most recent files
vi *(.om[1]^D)   # vi newest file ^D means switch off GLOB_DOTS
ls -l *(m4)      # list files modified exactly 4 days ago
ls -ltd *(mw3)   # list files 3 weeks old
echo *(m-1)      # files modified today
echo *(m0)       # files modified today
rm *.{aux,dvi,log,toc}(.N) # rm latex temp files N means no error msg is any file absent

print *(n:t)     # order by name strip directory
print **/*(On:t) # recursive reverse order by name, strip directory
print *.c(:r)    # strip suffix
ls **/*(.)       # only files no directories
-ld *(/)      # list only directories
FOO = (#i)foo ]]  # case insensitive matching
```

```
#oddities
fred=$((6**2 + 6)) # can do maths
print ${#path}     # length of "path" array
print ${#path[1]}  # length of first element in path array
ls fred{joe,sid}.pl
ls fred{09..13}.pl

# arrays
array=(~ /.zshenv ~ /.zshrc ~ /.zlogout)
% print ${array:t}
.zshenv .zshrc .zlogout

x="bu&^ *ck"                 # variable with mucky characters
print ${x//[^ [:alnum:]]/_}   # replace all non-alphanumerics with _


cp file ~ 1                  # where 1 is first entry in pushd stack
#zsh completion
startfilename<tab>           # will complete matching files anywhere in $PATH
startfilename<C-D>           # will list matching files anywhere in $PATH
#directory sizes
du -sk *(/)

ls * | grep foo | less
#to
ls * G foo L
#

#magic equals
vim =some_file                          # edits file anywhere in $PATH
ls =some_file                           # lists file anywhere in $PATH
#magic ** (recursion)
vim **/some_file                        # edits file under under current dir
# modifying more than one file (multios)
# writes ls results to file1 & file2 appends to filec
ls > file1 > file2 >> file3 | wc
```

Find file containing string ’printf’ in /usr/include.

```
$ zargs /usr/include/**/*.h −− grep printf /dev/null
```

A solution without zsh could look like:

```
$ find /usr/include -name \*.h −exec grep printf /dev/null {} ;
```

Create a directory structure based on an existing one.

```
$ dirs=(**/*(/))
$ cd −− $dest_root
$ mkdir −p −− $dirs
```

A solution without zsh could look like:

```
$ src=/usr/local
```

```
 $ dst=/opt
 $ cd "$src"
 $ find . -type d | cpio -pdmv "$dst"
```

Uncompress file and read it
less <(gzip -cd foo.gz)

A solution without zsh could look like:

```
 $ gzip -cd foo.gz && less foo
```

Print two files and sort them

```
 $ sort <f{oo,ubar}
```

A solution without zsh could look like:

```
 $ cat foo fubar | sort
```

Find files up from current directory and change permissions to ’700’.

```
 $ chmod 700 **/*(.)
```

A solution without zsh could look like:

```
 $ find . −type f −exec chmod 700 {} \;
```

List details of the executable ’foobar’.

```
 $ ls -l =foobar
```

A solution without zsh could look like:

```
 $ ls -l ‘which foobar‘
```

Small examples
´cd old new’ replaces ’old’ with ’new’ in directory-names.
´which -a cmd’ lists all occurences of ’cmd’ in $PATH.

OPTIONS

Navigation options

auto_cd (allow one to change to a directory by entering it as a command). auto_pushd (automatically append dirs to the push/pop list) pushd_ignore_dups (and don’t duplicate them)

Misc

no_hup (don’t send HUP signal to background jobs when exiting ZSH) print_exit_value (show a message with the exit code when a command returns with a non-zero exit code)

History options

hist_verify (let the user edit the command line after history expansion (e.g. !ls) instead of immediately running it)

Use the same history fi le for all sessions :
setopt SHARE_HISTORY

Privacy / Security

no_clobber (or set -C; prevent '>' redirection from truncating the given fi le if it already exists)

Spelling correction

correct (automatically correct the spelling of commands) correct_all (automatically correct the spelling of each word on the command line) dvorak (dvorak layout)

LINKS

The Z shell Homepage
http://www.zsh.org/

The Z shell FAQ
http://zsh.sunsite.dk/FAQ/

The Z shell wiki
http://www.zshwiki.org/

Mailinglistarchive
http://www.zsh.org/mla/

The Z shell reference-card (included in the zsh-lovers
debian-package) **http://zsh.sunsite.dk/Refcard/refcard.ps.gz**

Adam Spier's UNIX shells page
http://adamspiers.org/computing/shells/

The Single UNIX (R) Specifi cation, Version 2 - Shell Command Language Index
http://www.opengroup.org/onlinepubs/007908799/xcu/shellix.html

Zzappers Best of ZSH Tips
http://www.rayninfo.co.uk/tips/zshtips.html

The ZSH area on dotfi les.com
http://www.dotfi les.com/index.php3?app_id=4

Zsh Webpage by Christian Schneider
http://strcat.neessen.net/zsh/

The zsh-lovers webpage
http://grml.org/zsh/

IRC channel
#zsh at irc.freenode.org

AUTHORS

This manpage was written by Michael Prokop, Christian ´ strcat' Schneider and Matthias Kopfermann. But many ideas have been taken from zsh-geeks e.g. from the zsh-mailinglists (zsh-users and zsh-workers), google, newsgroups and the zsh-Wiki. Thanks for your cool and incredible tips. We learned much from you!

In alphabetic order:

Andrew ’zefram’ Main	- http://www.fysh.org/˜ zefram/
Barton E. Schaefer	- http://www.well.com/user/barts/
Matthias Kopfermann	- http://www.infodrom.north.de/˜ matthi/
Oliver Kiddle	- http://people.freenet.de/opk/
Paul Falstad	- http://www.falstad.com/
Peter Stephenson	- http://python.swan.ac.uk/˜ pypeters/
Richard Coleman	
Stéphane Chazelas	- http://stephane.chazelas.free.fr/
Sven Guckes	- http://www.guckes.net/
Sven Wischnowsky	- http://w9y.de/zsh/zshrc

SEE ALSO

Manpages of zsh:

zsh	Zsh overview (this section)
zshmisc	Anything not fitting into the other sections
zshexpn	Zsh command and parameter expansion
zshparam	Zsh parameters
zshoptions	Zsh options
zshbuiltins	Zsh built-in functions
zshzle	Zsh command line editing
zshcompwid	Zsh completion widgets
zshcompsys	Zsh completion system
zshcompctl	Zsh completion control
zshmodules	Zsh loadable modules
zshzftpsys	Zsh built-in FTP client
zshall	Meta-man page containing all of the above

Note: especially ’man zshcontrib’ covers very useful topics!

Book:
From Bash to Z Shell
by Oliver Kiddle, Jerry Peck and Peter Stephenson
ISBN: 1590593766

Also take a look at the section
LINKS
in this manpage.

BUGS

Probably. This manpage might be never complete. So please report bugs, feedback and suggestions to <zsh-lovers@michael-prokop.at>. Thank you!

COPYRIGHT