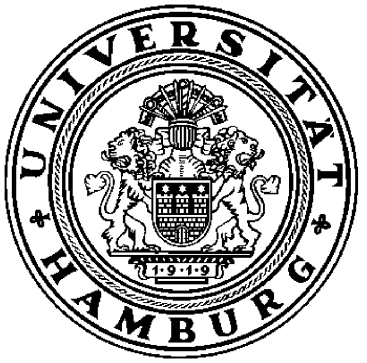
zsh and shell scripts

Michael Steder, 06.11.2006, oo get together



1 what are we talking about?

2 feel comfortable with the zsh

3 teaching an old dog new tricks (aka use the history)

4 using shell scripts

5 summary

what are we talking about?

a shell

- interface between OS and user

- **command line interpreter (CLI)**

- **in principle also**
  - X-Server
  - M$ Windows desktop

- **typical features of today's shells**
  - navigating in the directory tree
  - redirecting in-/output
  - expanding wild cards
  - job control
  - history
  - tab completion

small shell selection

- sh Bourne shell, 1977/78
- csh C shell, 1979
- ksh Korn shell
- bash Bourne again shell
- pdksh
- tcsh
- ash
- **zsh** **z shell, Paul Falstad, 1989** (**Z**hong **Sh**ao, userid: zsh)
- **....**

let's not forget about some retards...

- command.com MS-DOS
- cmd.exe Windows NT, 2000, XP, 2003

feel comfortable with the zsh

why do I need to quote ?

- some keys or characters have special meaning

e.g. “`- > < | / ^C $`”

- one might need/want to use them

- special characters between single quotes '...'

everything is quoted (e.g. `'\''` does NOT work)

- special characters between double quotes “...”

everything quoted - except `“, \, $,` and ´ (e.g. `“directory is $PWD”`)

the three levels of quoting

task: rename a file “`-|^C`” to “`filename`”

→ `mv -|^C filename`

1. parameters starting with “-” are options for mv **(application)**
   - quoting with “`./`” → `mv ./-|^C filename`
2. the pipe character “|” is used for redirecting I/O **(shell)**
   - quoting with “`\`” → `mv ./-\|^C filename`
3. Ctrl-C “^C” will be captured by the **terminal** and sends an interrupt to the shell
   - quoting with “`^V`” → `mv ./-|^V^C filename`

a simple quoting example

task: try to delete files starting with a space

“`rm ␣testfile`” → “`rm \ testfile`”

feel comfortable with the zsh

what's globbing ?

- **extension of wildcards to match multiple filenames**
- **very powerful in the zsh** (see also `man zshexpn`)
- **very brief here**

now let's benefit

- **`setopt extended_glob` enables additional features**
- **grep for “word” in all files except (~) compressed ones**

```
grep word *~(*.gz | *.bz | *.bz2 | *.zip | *.Z)
```

- **grep or list files recursivly using “`**`”**

```
print -l **/*.html
grep name **/*.txt
```

- **grep only in files containing a dot**

```
grep *(.)
```

- **list all files which are not (^) postscripts or pdf's**

```
ls ^(*.ps | *.pdf)
```

- **list all files changed up to one day before**

```
print -l *(a1)
```

- **change permissions of files everyone may write to**

```
chmod 640 *(w)
```

- **make life more efficient with aliases** (be careful...)

```
alias -g $k = "*~(*.gz | *.bz | *.bz2 | *.zip | *.Z)"
→ ls -d $k
unalias '$k'
```

!! mind the single quotes !! (or the alias will be interpreted)

feel comfortable with the zsh

redirects and pipes

- zsh supports redirecting to or from multiple files

```
ls > file1 > file2 > file3
less < file4 < file5
```

- zsh supports redirecting and piping at the same time

```
make > logfile | grep error
```

- cut out a part of the output

```
cat file | cut -d";" -f 1,3
```
(-d delimiter, -f fields)

```
cat /etc/resolv.conf | grep nameserver | cut -d" " -f 2
```

- piping to files within a shell script

```
cat >! $STEERFILE  << EOF
This text from here until End Of File will enter the file
formatted as written here....
EOF
```

- temporary files

e.g. show diff of two compressed files

```
diff <(zcat first.gz) <(second.gz)
```

act on no orders

- if no command is given, $READNULLCMD is used (default: `more`)

```
< file    →  more file
```

feel comfortable with the zsh

handle files within $PATH

- list or edit files with "=name"

```
ls -iL =xemacs
xemacs =passwd    (if /etc is part of $PATH)
```

wildcards

- list all files ending with .a or .b

```
ls *.[ab]
```

- list all files not ending with ' ~ '

```
ls *[^~]
```

change directories

- pushd is like cd, but stores current directory on a stack
- popd restores this later on
- pushd without specifying a directory stores the current dir

- same directory tree with different versions

use the old fashioned

```
msteder@h1trinidad:~/h1/30/a/b/c>cd ../../../../27/a/b/c/
```

or the zsh way

```
msteder@h1trinidad:~/h1/30/a/b/c>cd 30 27
msteder@h1trinidad:~/h1/27/a/b/c>
```

feel comfortable with the zsh

controll the jobs

`Ctrl-Z`	suspends a running job and returns to shell
`cmd &`	starts a job in the background
`jobs`	shows suspended and in background running jobs
`fg %2`	gets job 2 in the foreground
`bg %3`	puts job 3 in the background

get the right job

	get the...
`%%` or `%+`	last job
`%-`	job before last
`%2`	second job
`%xyz`	last job starting with `xyz`
`%?xyz`	last job containing `xyz`

or a bit shorter

`fg,%,%%` or `%+`	≙	`fg %%`
`bg`	≙	`bg %%`
`%2`	≙	`fg %2`
`%xyz, %?xyz`	≙	`fg %xyz, fg %?xyz`

teaching an old dog new tricks (aka use the history)

the shell history

- stores all shell commands in a history file `.history`

- ignores lines starting with a blank (`(un)setopt HIST_IGNORE_SPACE`)

use the shell history

common but *inefficient* ways

- scroll the whole terminal → cut and paste
- use “cursor up” to scroll through the list
- “`cat .history | grep command`” → cut and paste

if you pretend to be lazy – be lazy !

- get the correct line

`!!`	last command line (in history)
`!3`	third line
`!-2`	line before last
`!xyz`	last line starting with “xyz”
`!?xyz`	last line containing “xyz”

- get the correct parameter

`:*`	all parameters
`:^`	first parameter
`:$`	last parameter
`:2`	2nd parameter
`:2-4`	2nd - 4th parameter

teaching an old dog new tricks (aka use the history)

- modify history expansion

`:h`	remove last path component	(/h1/test → /h1)
`:t`	remove path of a filename	(/h1/test/file → file)
`:r`	remove last file extension	(/h1/test.tar.gz →:r :r → /h1/test)
`:e`	only keep the extension	(/h1/test.tgz → .tgz)
`:p`	only print result, don't execute	
`:s/a/b/`	replace “a” once by “b”	
`:gs/a/b`	replace all “a” by “b”	

let's become real sluggards...

- use short forms

`!^`	≙	`!!:^`
`!$`	≙	`!!:$`
`!:2`	≙	`!!:2`
`!*`	≙	`!!:*`
`^a^b`	≙	`!!:s/a/b`

or a bit more intuitively?

`Ctrl-R`	starts interactive search
`Meta-P`	searches for the text in the command line
`fc -l -10`	lists the last 10 commands
`fc`	opens last command to edit in $FCEDIT (default vi)
	echoes the command and executes

teaching an old dog new tricks (aka use the history)

efficient and excessive ...

```
tar cvf file-1.2.3woody4_i386.tar file
gzip -9v file-1.2.3woody4_i386.tar
mv file-1.2.3woody4_i386.tar.gz file-1.2.3woody4_i386.tgz
mkdir backup
mv file-1.2.3woody4_i386.tgz backup
mv backup Backup
cd Backup
```

```
tar cvf file-1.2.3woody4_i386.tar file
gzip -9v !:2
mv !$.gz !$:r:r.tgz
mkdir backup
mv !mv:$ !$
mv !$ !$:s/b/B
cd !$
```

- fault-tolerant but hard to read
- therefore better to use in shell scripts

using shell scripts

what's a shell script

- **an executable file containing**
  - linux commands
  - control structures
  - variables
  - comments
  - other shell scripts
- **reduces expenditure of time (for recurring tasks)**

how does a shell script look like

- **first line denominates the interpreting shell**

```
#! /bin/zsh
```

- **followed by the code**
- **commands are seperated by either a new line or a “;”**

comments

- **lines starting with a “#” are not interpreted** (except 1st line)
- **works also in the middle of a line** (for the rest of it)

variables

- **assignment**

```
name = X
```

- **access**

```
$name
```

using shell scripts

controll structures

- loop

while loops

```
while true; do echo -n .; sleep 1; done;
while [$VarA -le $VarB]; do echo -n .; sleep 1; done;
```

until loops

```
until false; do echo -n .; sleep 1; done;
```

for loops

```
for i in 1 2 3; echo $i; sleep 1; done;
for ((i=1;i<4;i++)); echo $i; sleep 1; done;
foreach flag (jpsi dstar ...); print $flag; end
foreach file ($(ls*.[ab])); print $file; end
```

- decide

```
if [ $flag = 'jpsi' ] ; then ; selection="NumJPsi>0" ; fi
if ((NumJPsi > 0 )) ; then ;... ; else ;...; fi
```

using command line parameters

- simply use "$n", where n is the position of the parameter

script.zsh:

```
#! /bin/zsh
print "Nice to see you, $1." #$1 is the 1st parameter
```

```
~$./script.zsh Shiraz
Nice to see you, Shiraz.
```

using shell scripts

in medias res

- small example for a shell script
- (( )) allows syntax similar to C
- calculates somehow funny...

```
#! /bin/zsh
# multiply.zsh
print
print "This program multiplies two numbers."
print "Usage: ./multiply.zsh a b,"
print "       where a and b are the numbers to multiply"
print
print $1" * "$2" = "$(($1*$2))
```

```
msteder@h1trinidad:~>./multiply.zsh 2.3 2

This program multiplies two numbers.
Usage: ./multiply.zsh a b,
       where a and b are the numbers to multiply

2.3 * 2 = 4.599999999999996
```

using shell scripts

replacement via character string indices

- replacing characters in strings
- negative indices count down from the end

```
#! /bin/zsh

a="i didn't do it."
print $a
a[1]='I'
a[-1]='. (Bart Simpson)'
print $a
```

```
msteder@h1trinidad:~>./test.zsh
i didn't do it.
I didn't do it. (Bart Simpson)
```

using shell scripts

associative arrays

- arrays with strings as indices
- syntax similar to maps in STL

```
#! /bin/zsh
#typeset is a very powerful formatting command
typeset -A ass_array; ass_array=(one    1
                                 two    2
                                 three  3
                                 four   4)

print $ass_array[one]
print $((ass_array[three]*ass_array[two]))
print ${(k)ass_array}  # returns list of keys
print ${(v)ass_array}  # returns list of values
```

```
msteder@h1trinidad:~>./test.zsh
1
6
four three two one
4 3 2 1
```

using shell scripts

unnamed temporary variables 1

- $(cmd) returns output of cmd execution
- [n] returns nth “word” of that output
- [n,m] returns nth - mth “word” of that output
- nested use possible

```
#! /bin/zsh

print date
print $(date)
print "          "${$( date )[4]}
#nested to retrieve time from date without seconds
print "          "${$( date )[4][1,5]}
```

```
msteder@h1trinidad:~>./test.zsh
date
Fr Nov  3 04:17:23 CET 2006
          04:17:23
          04:17
```

using shell scripts

unnamed temporary variables 2

- #String cuts of “String” at the beginning of the word

```
#! /bin/zsh

print ${${$( LC_ALL=de_DE /sbin/ifconfig eth0 )[7]}}
print ${${$( LC_ALL=de_DE /sbin/ifconfig eth0 )[7]}#Adresse:}
```

```
msteder@h1trinidad:~>./test.zsh
Adresse:131.169.103.233
131.169.103.233
```

using shell scripts

unnamed temporary variables 3

- get a specific line out of a file

```
#! /bin/zsh
#This is the 2nd line of test.zsh

#print 2nd line from file test.zsh
print -l ${"$( < test.zsh )"[(f)2]}
print

#print 1st line containing “root”
print -l ${"$( < /etc/passwd )"[(fr)*root*]}

#print 1st line containing “root” cut off userID
print -l ${"$( < /etc/passwd )"[(fr)*root*]#root:}
```

```
msteder@h1trinidad:~>./test.zsh
#This is the 2nd line of test.zsh

root:x:0:0:root:/root:/bin/bash
x:0:0:root:/root:/bin/bash
```

using shell scripts

calculating with the z shell

- the mathfunc module offers a lot of functions
- load module with `zmodload zsh/mathfunc`

```
#! /bin/zsh

zmodload zsh/mathfunc
print "sin(1+2) = "$((sin(1+2)))
print "sqrt(2) = "$((sqrt(2)))
print "2^2 = "$((2**2))
```

```
msteder@h1trinidad:~>./test.zsh
sin(1+2) = 0.1411200080598672
sqrt(2) = 1.4142135623730951
2^2 = 4
```

- output in different bases

  `[#base]` prints base and number

  `[##base]` prints only number

```
for ((i=1; i<20; i++)){ print $(([#10] i))}   #dec
          -"-           { print $(([#16] i))}   #hex
          -"-           { print $(([##16] i))} #hex only num
```

using shell scripts

testing files

- various tests on file conditions can be made

	returns true, if file...
-e	exists
-d	is a directory
-g	has setgid bit set
-h	is symbolic link
-k	has sticky bit set
-r	is readable
-s	has size > 0
-w	is writeable
-x	is executable
-O	is owned by UID
-G	is owned by GID

two argument test `[[a test b]]`

-nt	a newer than b
-ot	a older than b
-....	

```
#! /bin/zsh
if test -e test.zsh
then
print "test.zsh exists";
else
print "test.zsh missing";
fi
```

```
msteder@h1trinidad:~>./test.zsh
test.zsh exists
```

summary

overview over zsh features

- not really brief
- nevertheless very fragmentary
- see `H1Tools/submit_oosubsetuser.sh` to learn about the use of scripts

many commands make life more comfortable

- on the shell
- in shell scripts

all other script languages can be used

- Perl, Python, Ruby, ...
- not mentioned here
- listen to Mira (regExp in perl, 27.11.) or Christian (perl introduction, 18.12.)

more features of the zsh

- a free programmable TAB completion
- compatible with bash, ksh, tcsh
- very flexible prompt
- use ~/.zshrc to configure your personal zsh

Love your shell and your shell will love you...