

Computing at DESY Zeuthen

An Introduction - Part II

Stephan Wiesand
Summer Students Lecture
Zeuthen, 2009-07-23

> Part I

- computing environment
- policies
- resources
 - > desktop PCs (linux)
 - > login hosts & farms
 - > storage, AFS basics
- getting started
 - > basic shell usage
 - > email, printing
 - > application software

> Part II

- advanced shell usage
 - > options, aliases
 - > scripting
 - > pipelines, I/O redirection
- more about AFS
 - > ACLs
- using the batch farm
- building software
 - > compiling & linking
 - > make, debugging

Environment Variables

- > the shell has variables:
 - `my_var="some_value"`
 - > no space allowed around "="
 - `echo $my_var`
 - > dereferencing by prepending a "\$"
 - > more generally: `${my_var}`
- > shell variables can be **exported**:
 - `export my_var`
 - `export my_var="some_value"`
- > exported variables are **available to child processes**
 - and called "**environment variables**"



Commonly Used Variables

> PATH

- a list of directories, separated by colons (":")
- where the shell looks for **commands**

> LD_LIBRARY_PATH

- where the dynamic loader looks for **shared libraries**

> PRINTER and LPDEST

- where your printjobs go by default

> **env** prints the **complete environment**

> **echo \$<var>** prints a **single variable**



Where to Set the Variables

> ~/.zprofile

- variables set and exported here are available to all your processes
- do **NOT** change **PATH** or **LD_LIBRARY_PATH** here
 - > unless you really really know what you're doing
 - > no references to external sites
 - may **slow down** most **everything** considerably
 - > note: **ini** changes both => **NO ini** in ~/.zprofile or ~/.zshrc

> scripts

- generally the right place
- > generally try to **avoid using LD_LIBRARY_PATH**



Globbing

> Unix jargon for **wildcards**

- `ls -l *.c` -> all .c files
- `ls -l *.[chf]` -> all .c or .h or .f files
- `ls -ld /usr/?bin` -> /usr/sbin
- `echo /usr/bin/a*k` -> /usr/bin/awk

> expansion is **done by the shell, not the command**

- `scp pub3:/tmp/mydir/*.c ~/`

> **does not work** as (often) expected

> because **globbing is done locally**

> use **single** quotes to **prevent any expansion**

- `scp 'pub3:/tmp/mydir/*.c' ~/` works
- double quotes still expand variables



Command Aliases

> `alias my_command='echo foo'`

- `my_command` will print "foo"

> `alias command2='my_command; echo "bar"'`

- `command2` will print 2 lines: "foo" and "bar"
- note the **semicolon separates commands**:

> `cd /tmp; ls`

> aliases can be set in `~/.zshrc`

- read by all **interactive shells**

> a plain `alias` will print all defined aliases



I/O Redirection

> processes have three I/O channels by default

- `stdin` reads input
- `stdout` prints normal output
- `stderr` prints error messages

> `ls > list.txt`

- `redirects stdout` of `ls` into file `list.txt`
- errors are still printed to terminal

> `ls > list.txt 2>&1`

- `redirects stderr to stdout`, and `both to list.txt`
- `=>` also errors go into `list.txt`



Input Redirection, Pipes

> `echo '3*4' > infile; bc < infile`

- prints "12"
- bc is the "binary calculator"
- "<" redirects stdin

> `ls -l /usr/bin | less`

- | connects stdout of ls with stdin of less
- called a "pipe"
- use `2>&1 |` to pipe stdout and stderr, or short: `| &`

> I/O redirection does not work for commands using the terminal in "raw" mode

- `passwd < my_passwd.txt` does not work (which is good)



Conditionals

> `command1 && command2`

- executes `command2` if and only if `command1` succeeds
- commands return an integer to their parent process
- 0 signals success
- anything else signals failure
- return value of last command is in variable `$?`

> `command1 || command2`

- executes `command2` if and only if `command1` fails

> `command1 && echo "ok" || echo "failed"`



Conditionals continued

```
> if test -e /some/file
then
    do_something
else
    echo "/some/file is missing"; exit 1
fi
```

- is another way to do this

- `test` is `/usr/bin/test`

 - > returns 0 or 1, depending on test result

 - > `test -e <file>` tests whether file exists

- can also be written `if [-e /some/file]; then`

> **interactive shell will prompt nicely** if you hit return after a line opening an if clause



Loops

> `for i in 1 2 3 4 5; do echo $i ; done`

- prints 5 lines: "1", "2",...
- `for i in {1..5}; do echo $i; done` is the same

> `for f in *.c ; do cp $f $f.BAK ; done`

- creates copies of all c-files in current directory
- effectively: `cp file1.c file1.c.BAK ; cp ...`

> `for f in *.c ; do cp $f `basename $f .c`_BAK.c ; done`

- `basename <file> <suffix>` strips suffix off name
- the `backticks` substitute the output of their command
- effectively does `cp file1.c file1_BAK.c ; ...`

> `while ["$finished" = 0]; do run_my_cmd ; done`

- yes, there's awhile loop too



Shell Scripts

- > recipe for creating a shell script:
 - 1) create a file with a first line `#!/bin/zsh`
 - or, maybe, `#!/bin/sh`
 - 2) fill it with shell commands
 - 3) make it executable with `chmod +x`
- > this script can be called like any other command
- > arguments are available as `$1`, `$2`, ... in scripts
- > if you have some software that needs a special `LD_LIBRARY_PATH`, write a wrapper script and place it into `~/bin`



Wrapper Prototype

```
#!/bin/zsh  
  
export LD_LIBRARY_PATH=/afs/cern.ch/atlas/libs  
  
some_command "$@"
```

- > some_command will be executed with the right LD_LIBRARY_PATH in its environment
- > will not affect anything else
- > “\$@” expands to the list of all parameters passed to the script



Summary: the Shell

- > a very powerful tool worth learning
- > for more information, see
 - the [zsh man/info pages](#)
 - the [bourneint.ps](#) document (use google to find it)
- > caveats:
 - what was shown works for the [bourne shell family](#)
 - > zsh, ksh, bash, sh
 - > there are minor differences between those
 - there is also a [csh family](#) with a very [different syntax](#)
 - > csh, tcsh
- > NB: even more powerful scripting: perl, python



More about AFS

> AFS is a global filesystem

- segmented into "cells", path: `/afs/<cell>/...`
 - > NB: `/bin/pwd` (not just `pwd`) shows real current directory
- DESY Zeuthen cell: `ifh.de`
- DESY Hamburg cell: `desy.de`
- CERN cell: `cern.ch`

> some of its features:

- good security: valid token needed for access
- data replication (readonly)
- data relocation (readwrite, transparent to clients!)
- persistent client cache



AFS Client Cache

- > the client maintains a local cache
 - persistent (still available after reboot)
 - read-write
- > local changes to a file are flushed to the server when the file is closed
- > while you edit a file, the authoritative copy resides locally on your PC
 - PCs should be shut down cleanly
 - > do NOT use the power or reset buttons
- > a file being changed on another host may appear empty or unchanged, until flushed there



AFS Quotas

- > AFS space is handled in chunks called **volumes**
 - your **home directory** is one **volume**
 - your `~/.OldFiles` snapshot is another volume
- > each **volume** has an **associated quota**
- > **`fs listquota <path>`** shows
 - the quota (**maximum** amount of data allowed)
 - the **current** usage
 - > you should **stay below 95%**
 - is another way to find out whether a dir is in AFS
 - **`~/.OldFiles` does not count** for `fs listquota` ~



AFS Permissions: ACLs

- > AFS permission system is different:
 - traditional Unix filesystem has **r**ead, **w**rite, **e**xecute
 - AFS has
 - > **r**ead, **w**rite, **i**nsert, **d**eleate,
 - > **l**ookup, **l**ock, **a**dministrat
 - all these are **per directory**
 - traditional mode bits are mostly ignored
 - but the **x** bit retains its meaning
 - an **ACL** is a list of pairs: (<who>, <mode>)
 - > **who**: a **u**ser, or a **g**roup
 - > **mode**: a list of **b**its, like **r****w****i****d**



Examining ACLs

- > is also done with the **fs** command:
 - **fs listacl <path>** shows ACL of a directory
- > **fs listacl ~** should show
 - **system:administrators rlidwka**
 - > the sysadmins can do anything
 - **system:anyuser l**
 - > any user worldwide (!) can lookup files (follow symlinks)
 - **<user> rlidwka**
 - > you can do anything as well
- > do **NOT** change the ACL of your ~



Changing ACLs

> `fs setacl <path> <who> <mode>`

- handy shortcuts for mode:

> `read` for `rl`

> `write` for `rlidwk`

> `all` for `rlidwka` (careful!)

> `none` for `""`

- `fs setacl ~/code group:amanda read`

> make `~/code` readable for amanda group

- `fs setacl ~/code <user> write`

> allow a colleague to do anything but change the ACL

> good for collaborative work

> but better done in group space, not home directory



The AFS sysname

- > a per-host property

- 32-bit Scientific Linux 5 in Zeuthen: i586_rhel50
- 64-bit: amd64_rhel50
- SL4: i386_linux26
- Solaris 8 on SPARC: sun4x_58

- > `fs sysname` shows the value for a host

- > a path component `@sys` is replaced by the sysname

- only in AFS

- typical usage:

- > set a link `.../bin -> .../@sys/bin`

- > call `.../bin/command` to get the right binary automatically

- > our systems have a `sysname list` for compatibility



Summary: AFS

- > AFS is the **best filesystem we have**
 - is also true for the **hardware storing homedirs**
 - > please do not waste the space, it's **precious**
- > AFS is **best for collaborative work**
 - NB: **~/public/www** is available as `http://www-zeuthen.desy.de/~<user>`
 - note **~/public is really public**
- > AFS space is the **right place for**
 - **valuable** files (source code) - if backed up
 - **confidential** files (CV, saved mails, ...)
- > AFS is not particularly fast



Using the Batch Farm

- > need to perform some serious calculation?
- > PCs usually not powerful, limited RAM
- > public login systems (pubs, lx64, ..) are NOT to be used for actual compute jobs
 - development & test only!
- > WGS are limited, and not meant to be abused as well
- > farm has hundreds of fast CPUs w/ plenty of RAM
 - usage:
 - 1) split task into jobs
 - 2) script them
 - 3) submit the job scripts
 - very powerful resource
 - > can cause serious problems if used without care



Example Batch Job

```
#!/bin/zsh
#$ -S /bin/zsh
#$ -l h_cpu=04:30:00
#$ -l h_vmem=350M
#$ -j y
#$ -m ae
#$ -N my_job

hostname; date; env

cd $TMPDIR
cp ../infile .
[ $? -eq 0 ] || cp ../infile .

do_the_work $SGE_TASK_ID

cp outfile /afs/...
[ $? -eq 0 ] || ...
```

otherwise the default shell would be used
the cpu time for this job
the maximum memory usage of this job
stderr and stdout are merged
send mail on job's end and abort
the name of the job

some info we want in the stdout file

always \$TMPDIR, NOT /tmp !
fetch input
retry if that failed, repeat...

run the actual job, output to \$TMPDIR

store the output file
retry if that failed...

- > `#$...` is interpreted by the batch system, the rest is an ordinary shell script
- > submit with `qsub -cwd -t 1:32 my_job.sh`



Batch: Commands

> **qsub**

- submit a job (array)

> **qstat**

- shows running/waiting jobs

> **qhost**

- shows status of execution hosts



Batch: Precautions

- > make sure you have sufficient **filesystem quota**
 - for all job output
- > **be nice to filesystems**
 - avoid jobs **writing the same file**
 - avoid too many jobs **working in the same directory**
 - > avoid writing too much to stdout/err (home or submit directory)
 - avoid too many jobs **using the same filesystem**
 - usually, **transfer data at beginning/end of job** only
 - > most of the time, work on the local disk, in \$TMPDIR
- > avoid **mass failures**, they cause **mail storms**
 - always send a few test jobs first



Building Software

- > if your project is small & simple, it's easy:
 - `<compiler> -o my_prog <source1> ...`
 - > `gcc -o my_prog *.c`
- > for more complicated projects:
 - two steps:
 - > `compile` source files into object files
 - > `link` object files + libraries to build the executable
 - `shared libraries may need some extra attention`
 - commonly done using `make`
 - > recompile only files that changed
 - > build according to `rules` defined in a `Makefile`



The test Trap

> has this happened to you?

- you have a file test.c, and run `gcc -o test test.c`
- you run test, and nothing happens
 - > there's a /usr/bin/test command
 - > /usr/bin is searched before . (PATH variable)

> another common case, with the same reason:

- a group has some standard programme, in your PATH
- you build a modified version and run it (you believe)
- your changes seem not to make any difference...

> make it a habit to use `./<command>`



Compilers Available (Linux)

- > default: `gcc`, `gfortran (g77)`, `g++` (Solaris: also `cc`, `f77`, `CC`)
 - use these unless there's a good reason not to
 - > could be: performance, fortran 90/95
 - on SL5, the native fortran compiler is `gfortran`
 - > `g77` is from the `gcc34` suite (backward compatibility)
- > `intel` compiler:
 - `ifort`, `icc`, `icpc`
 - different versions available via `ini`
- > `portland group` compiler
 - use `ini -v pgi` (also before running your programs)



Common Compilation Options

> -c

- only compile, do not link

> -g

- add debugging information to output file

> -O

- optimize (may be incompatible with -g)
- often available as -O1 or -O2 or ...

> -o <filename>

- change the name of the output file

> -I<path> [-I<path2> ...]

- prepend paths to search path for includes



Linking

- > always **use the compiler to link**
 - do not call the linker directly
 - the compiler knows about language specific libraries
- > common **options**:
 - **-L<path>**
 - > prepend path to search path for libraries
 - **-l<some_lib>**
 - > link against **libsome_lib.so**
 - if available, the **shared library** is preferred
 - > or against **libsome_lib.a**
 - otherwise, the **static library** is used



A Complete Example

> let's suppose you

- have **two fortran files**:

 - > main.f and fit.f

- and have to **link against cernlib**:

 - > libkernlib.a libpacklib.a libmathlib.a

 - > found in /cern/pro/lib

> g77 -c -g -o main.o main.f

> g77 -c -g -o fit.o fit.f

> g77 -o my_fit_prog main.o fit.o \
-L/cern/pro/lib -lkernlib -lmathlib \
-lpacklib



About Mixing Languages

- > mixing C and C++ is rather simple:
 - declare interfaces **extern "C"** in C++
 - use the C++ compiler for linking
- > mixing C/C++ with FORTRAN isn't:
 - fortran symbols usually have an "_" appended
 - > C's symbol for function some_func() is `some_func`
 - > FORTRAN's is `some_func_` or even `some_func__`
 - > g77 options: `-funderscoring`, `-fno-second-underscore`
 - a tool for interfacing: `cfortran.h`
 - use g++ for linking, add `-lg2c` (maybe more)



Using Shared Libraries

> advantages over static libraries:

- faster linking
- smaller executables
- less RAM needed if multiple programmes using the same library are running on the same system

> problem:

- all shared libs needed for running must be found at run time

> `ldd <executable>` shows the ones actually found

- "not found" for one means no go at all



How programmes find shared libs

- > sorted by precedence, this is determined by:
 - system's **dynamic linker configuration**
 - a list of **search paths** can be recorded at compile time
 - **LD_LIBRARY_PATH** in environment (**avoid!**)
- > recording a list of paths can be achieved by
 - an environment variable **LD_RUN_PATH**, or
 - a **-rpath <path> [...]** argument to the linker
 - > using the compiler for linking, this must be written as **-Wl,-rpath,<path> [-Wl,-rpath,<path2> ...]**
 - > in some cases, **-rpath-link** is needed as well
 - use one of these methods if possible



The make Tool

- > make is **not a script processor**
- > Makefiles are **not scripts**
 - typically not processed top to bottom
- > **make is a tool to create files**
 - typically from other files (-> **dependencies**)
 - according to **rules**
 - rules are defined in the **Makefile**
- > prefer **GNU make** (non-Linux: typically available as **gmake**)
 - available on all relevant platforms
 - generally superior to vendor's make



Our Example Using make

```
# the Makefile

main.o: main.f
    g77 -c -g -o main.o main.f

fit.o: fit.f
    g77 -c -g -o fit.o fit.f

my_fit_prog: main.o fit.o
    g77 -o my_fit_prog main.o fit.o \
        -L/cern/pro/lib -lkernlib -lpacklib -lmathlib
```

← a Tab character!

- > **make my_fit_prog** will now do the job
- > is already **better than a script**
 - recompiles only changed files

Make Targets & Rules

- > our make file has three targets
 - main.o, fit.o, my_fit_prog
 - **<target>: <dependencies>**
 - > read ":" as "depends on"
 - > empty dependencies are ok
- > **make <target>** means: create the file <target>
- > a simple **make** means: **make <topmost target>**
- > the lines after the target definition tell make **how to create the file** (must start with a **tab**)
 - together, this is called a **rule**



Our Example with a Default Target

```
# the Makefile

all: my_fit_prog

main.o: main.f
    g77 -c -g -o main.o main.f

fit.o: fit.f
    g77 -c -g -o fit.o fit.f

my_fit_prog: main.o fit.o
    g77 -o my_fit_prog main.o fit.o \
        -L/cern/pro/lib -lkernlib -lpacklib -lmathlib
```

- now a simple make will create my_fit_prog
 - unless the file "all" exists

make Variables

```
FC:=g77
FCOPTS:=-c -g
LIBS:=-L/cern/pro/lib -lkernlib -lpacklib -lmathlib

all: my_fit_prog

main.o: main.f
    $(FC) $(FCOPTS) -o main.o main.f

fit.o: fit.f
    $(FC) $(FCOPTS) -o fit.o fit.f

my_fit_prog: main.o fit.o
    g77 -o my_fit_prog main.o fit.o $(LIBS)
```



make Variables

- > can be set in the Makefile with
 - `=` evaluated recursively
 - `:=` no recursion (can be much faster - use this)
- > can also come from the **environment or command line**
- > **make FC=ifort** would use the intel compiler instead
- > useful special variables:
 - `$@`
 - > the target file of a rule
 - `$<`
 - > the input file(s) of a rule



Special make Variables

```
FC:=g77
FCOPTS:=-c -g
LIBS:=-L/cern/pro/lib -lkernlib -lpacklib -lmathlib
OBJECTS:=main.o fit.o

all: my_fit_prog

main.o: main.f
    $(FC) $(FCOPTS) -o $@ $<

fit.o: fit.f
    $(FC) $(FCOPTS) -o $@ $<

my_fit_prog: $(OBJECTS)
    $(FC) -o $@ $(OBJECTS) $(LIBS)
```



Generic Rules

```
FC:=g77
FCOPTS:=-c -g
LIBS:=-L/cern/pro/lib -lkernlib -lpacklib -lmathlib
OBJECTS:=main.o fit.o

all: my_fit_prog

# get rid of all builtin default rules
.SUFFIXES:

# how to compile fortran source files
%.o: %.f
    $(FC) $(FCOPTS) -o $@ $<

my_fit_prog: $(OBJECTS)
    $(FC) -o $@ $(OBJECTS) $(LIBS)
```



Summary: make

- > very powerful tool
- > prefer it over scripts for building
- > can do much more
 - additional dependencies (on include files...)
 - > can even be done automatically (but not trivial)
 - substitute shell command output
 - > use xxx-config commands to get libs, include paths
 - more and more packages have one (ROOT, cernlib, ...)
 - perform transformations on variable content...
- > consult make's info pages for more information



Debugging Your Software

- > **compile** all source files to be debugged **with -g**
 - compile without -O, or result may be confusing
- > for gcc & friends, the debugger is **gdb**
 - other compilers may need others
- > gdb itself is not very convenient to use
- > convenient **frontends**:
 - **(x)emacs** - use M-x gdb
 - > very usable, but takes some getting used to
 - **ddd**
 - > GUI, very easy to use



gdb Commands

- > **step** single step to next source line
- > **next** like step, not stepping into subroutines
- > **break** set a breakpoint (at file:line or a routine)
- > **cont** continue running until finished or breakpoint
- > **print** print a variable's content
- > **display** keep printing a variable's content
- > **watch** stop execution when a variable changes
 - dynamic breakpoints
- > many more ...



Appendix A

> Remember:

- always have a valid AFS token, and some space left in ~
- think thrice about what you store where
- don't abuse the public login systems nor your group's WGS
- mail problems/requests to uco-zn@desy.de
 - > include as much information as possible

> Some URLs (useful, but maybe hard to find):

- <http://dvinfo.ifh.de>
- <http://dv-zeuthen.desy.de/services/mail>
- http://www-it.desy.de/support/help/uco_documentation/afs.html.en
- <http://www-zeuthen.desy.de/~wiesand/intro/>



> Again: Have a pleasant and successful stay here at DESY Zeuthen!

> Questions ?