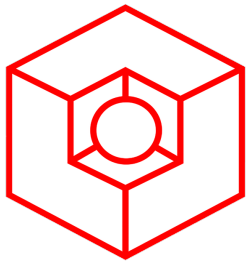


BremHLR

Kompetenzzentrum für Höchstleistungsrechnen Bremen

Using the Unix Shell

Lars Nerger (BremHLR & Alfred Wegener Institute)

Universität Bremen

CONSTRUCTOR UNIVERSITY

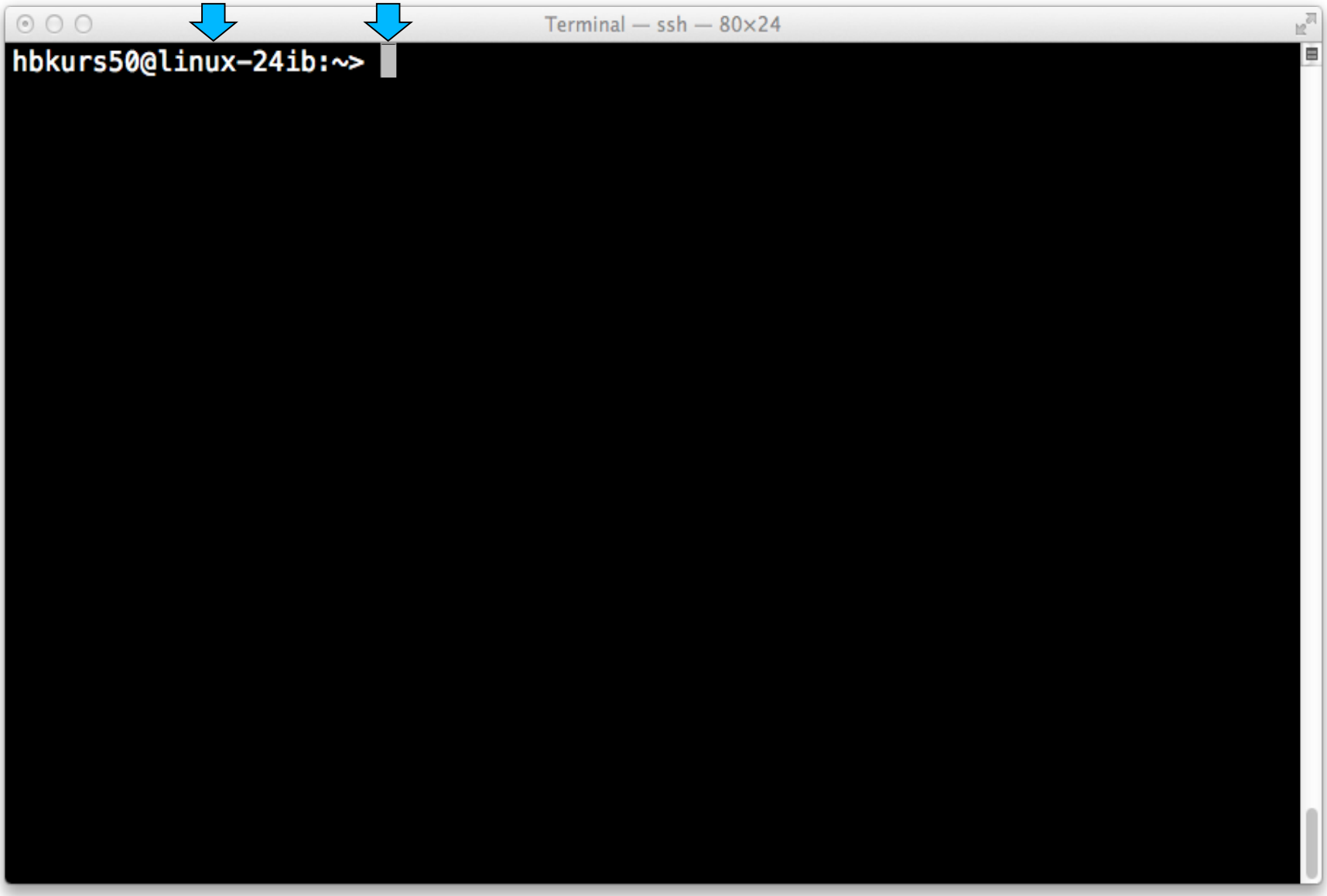
Hochschule Bremerhaven

NATIONALES HOCHLEISTUNGS RECHNEN

```
hbkurs50@linux-24ib:~>
hbkurs50@linux-24ib:~>
hbkurs50@linux-24ib:~>
hbkurs50@linux-24ib:~>
hbkurs50@linux-24ib:~>
hbkurs50@linux-24ib:~> The shell is older than a 'window'.
```

A shell window

The ‘prompt’ ‘curser’



The shell is just a program – several choices: **bash**, tcsh, sh, csh, ksh

Why using a ‘shell’?

- We aim at using a compute server or supercomputer
- Compute servers
  - don’t have a monitor attached
  - are only accessed remotely
  - usually don’t run a graphical interface
- Thus
  - one connects with ‘ssh’ (secure shell)
  - one uses a ‘shell’ to work on the supercomputer (compiling, submission of compute jobs)

Why using a ‘shell’? (II)

- apart from the restrictions on a compute server …

- A ‘shell’ is a powerful tool!
  - for handling files and directories
    - copying, moving, renaming, comparing, running programs, …
  - can be faster than a graphical tool
  - is available on any Unix/Linux/MacOS computer

(Windows had a powerful shell once ago, but different syntax)

Unfortunately:

The ‘shell’ doesn’t tell you which commands are available

Files

Files are identified by

- Name
  - E.g. **notes.txt** or **program.c**
  - Name extension can be associated with application (but it's not enforced)
- Path
  - directory in which the file resides
  - E.g. **/home/hzkurs50**

The file system

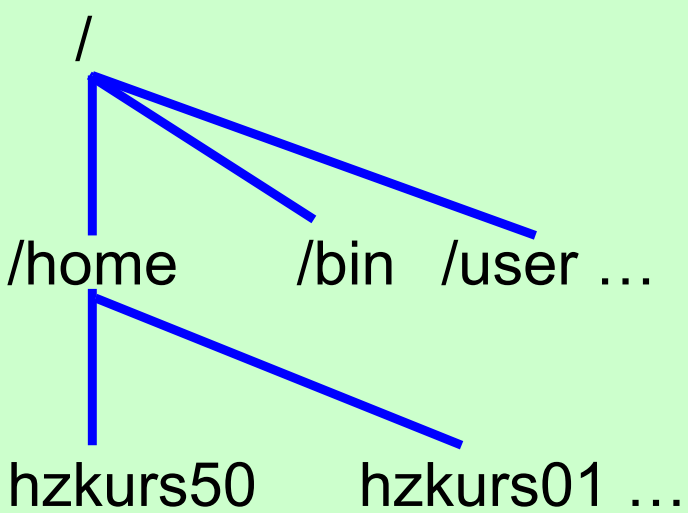
Built out of ***directories***

First: The root directory **/**

- Directories can contain directories
  - Gives the ***directory tree***
  - Names within a directory must be unique
  - Unix names are case dependent:

    file.txt ≠ File.txt

Directory tree



Navigating the file system

When we open (start) a shell

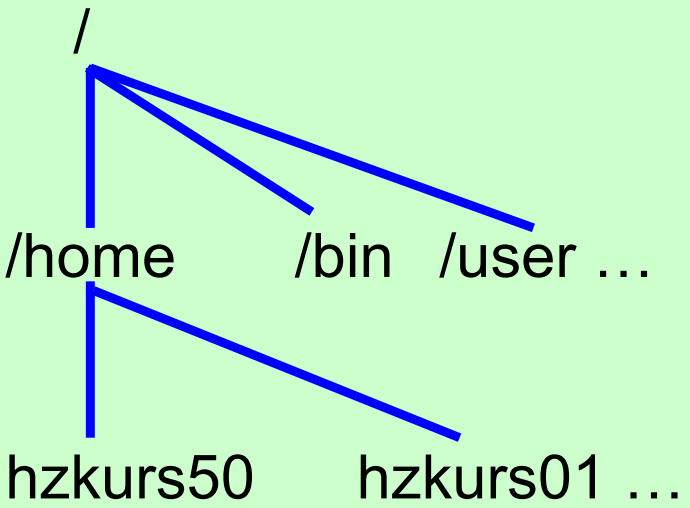
- We are in the ***home directory*** (e.g. /home/hzkurs50)

Some commands:

`pwd` check name of current directory

`cd DIRNAME` change into directory ‘DIRNAME’

Directory tree



Always send a command to the shell with ‘RETURN’

Specialties

`cd` without ‘DIRNAME’ changes into home directory

`cd ~` also changes into home directory

`cd ..` change into parent directory

See contents of a directory - *ls*

`ls` shows contents of a directory ('list'),

e.g.

```
hbkurs50@linux-24ib:~> ls
bin      Documents  example.sh  opt      Public     test
Desktop  Downloads  Music       Pictures Templates  Videos
hbkurs50@linux-24ib:~>
```

- Colors show the file type (directories, executables)
  (a common configuration)

- On some Unix/Linux systems you only see

```
hbkurs50@linux-24ib:~> ls
bin      Documents  example.sh  opt      Public     test
Desktop  Downloads  Music       Pictures Templates  Videos
hbkurs50@linux-24ib:~>
```

Options of ls

`-F` Add a file identification (/ for directory, * for executable programs)

`-s` show size
(for directories this is not the sum of the file sizes inside)

`-a` show also hidden files (with '.' at the beginning)

`-l` long listing format (showing details)

`-t` order files by time instead of name (young to old)

`-S` order files by size (large to small)

`-r` reverse the order

`-G` show file types with colors (also `--color`)

Options can be combined

```
ls -ltr
```

ls -ltr

```
hbkurs50@linux-24ib:~> ls -ltr
total 48
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Videos
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Templates
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Public
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Pictures
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Music
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Downloads
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:48 Documents
drwxr-xr-x 2 hbkurs50 workshop 4096 Dec  5 11:49 Desktop
drwxr-x--- 3 hbkurs50 workshop 4096 Dec  6 14:46 opt
drwxr-x--- 2 hbkurs50 workshop 4096 Dec  9 11:18 bin
drwxr-x--- 2 hbkurs50 workshop 4096 Dec  9 11:22 test
-rwxr----- 1 hbkurs50 workshop  134 Dec 13 11:16 example.sh
hbkurs50@linux-24ib:~>
```

type	rights	owner (user & group)	size	creation/ modification date	name

Effective use of ls

You don’t need to ‘cd’ into a directory to look into it!

`ls`	display contents of current directory
`ls ~`	display contents of *home* directory (from whereever you are)
`ls bin`	display contents of my subdirectory ‘bin’
`ls /bin`	display contents of system directory ‘/bin’
`ls /`	display contents of *root* directory

You can create an ‘alias’ for your favorite command

```
alias lsl="ls -ltr"
```

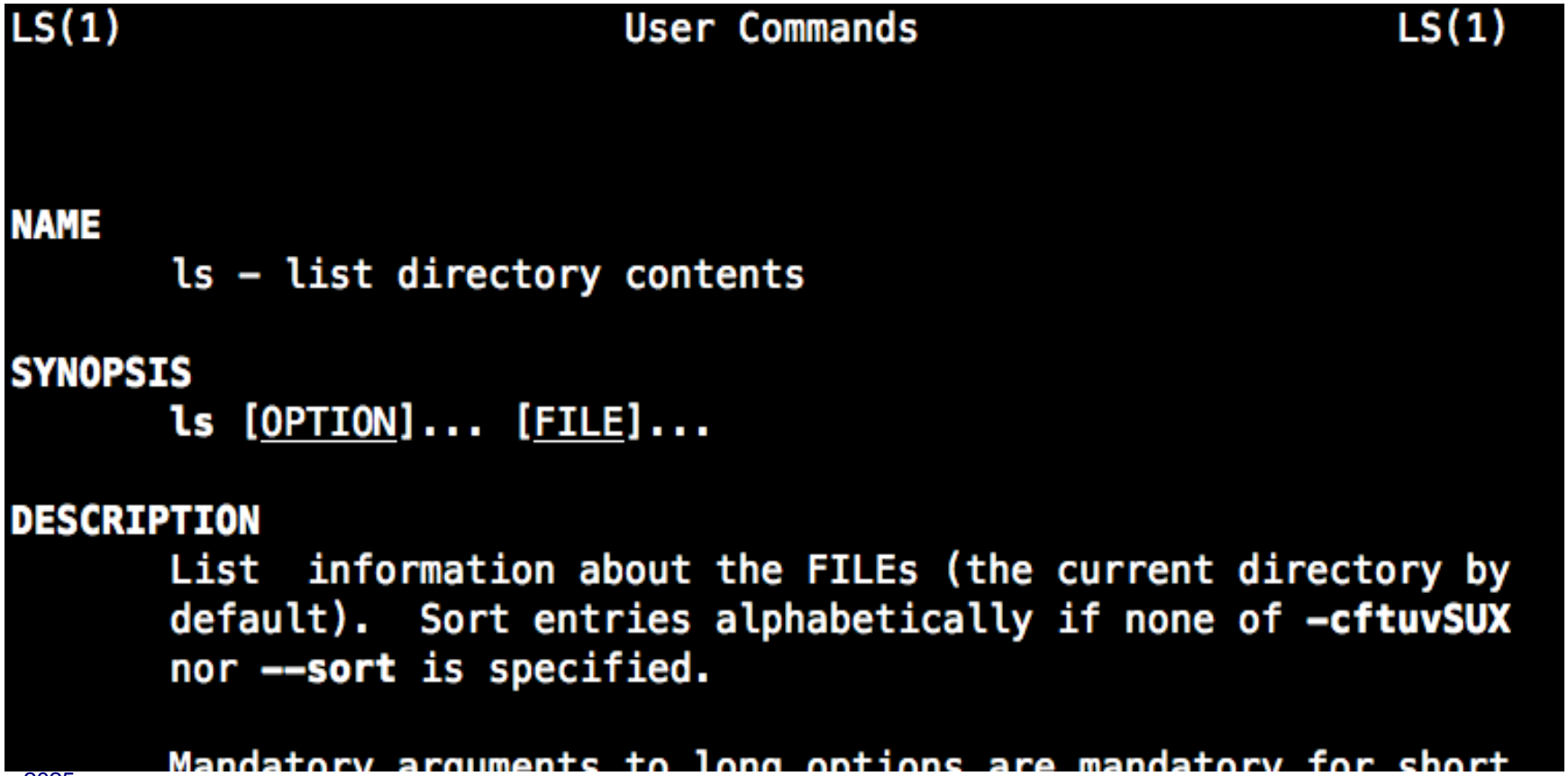
Information about commands - man

To obtain detailed information about a command, use

```
man COMMAND
```

e.g. ‘man ls’ (type 1 of you are asked what manual page to show)

Exit `man` by typing ‘q’



Starting an editor

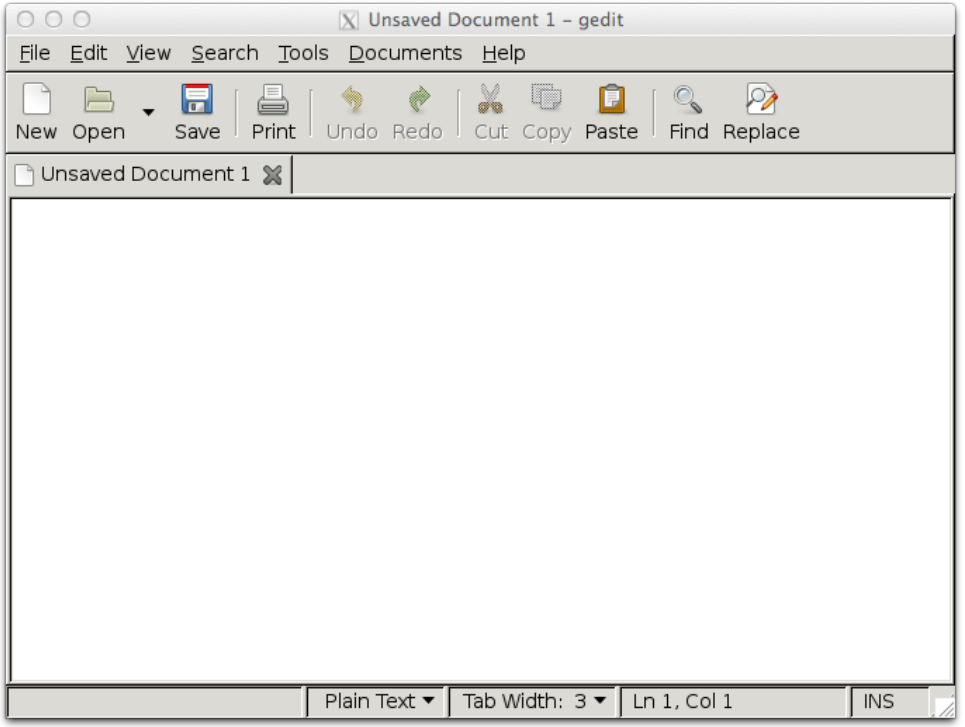
`cd` and `ls` allow to move around in the file system and see the content of directories

Later in the parallelization workshop, you want to create and edit files

- Convenient with an editor (graphical not in the shell):

Start the editor, e.g. *kate*

```
hbkurs50@linux-24ib:~> kate
```



Creating directories

For organizing your files …

Create the directory ‘course’:

```
mkdir course
```

It is created as a sub-directory of current directory.

To create the directory as sub-directory of HOME, from any directory:

```
mkdir ~/course
```

You can’t do

```
mkdir /course
```

Removing files and directories – *rm/rmdir*

To remove a single file, use

```
rm FILENAME
```

There are various options for `rm`

- `-i` often active by default: Confirm every removal
- `-r` recursive (can be dangerous)
- `-f` forced removal (very dangerous in combination with `-r`)

Remove a directory (if it is empty)

```
rmdir NAME
```

`rm/rmdir` have no wastebin: *removed* = *gone*

Copying and moving files – *cp/mv*

Create a copy of a file

```
cp FILENAME NEW_FILENAME
```

Obviously, the file named FILENAME has to exist.

You can create the copy in some other directory:

```
cp FILENAME /tmp/NEW_FILENAME
```

Moving files into another directory

```
mv FILENAME DIRNAME/FILENAME
```

Renaming files

```
mv FILENAME NEW_FILENAME
```

Looking into files – without editor

Look into text files – e.g. source code - without starting an editor:

```
cat example.sh
```

This lists the whole file – not useful for long files

For long files

```
less example.sh
```

or

```
more example.sh
```

`more` is really old – see man page:

```
more - file perusal filter for crt viewing
```

`less` is a bit younger - man page of less:

```
less - opposite of more
```

☺

Comparing files - *diff*

Display differences between two files

```
diff FILE1 FILE2
```

Differences are displayed like

Example

File_1.txt:

```
This is the first file!
This is the second line
```

File_2.txt:

```
This is the second file.
This is the second line
```

```
diff File_1.txt File_2.txt
```

```
1c1
< This is the first file!
---
> This is the second File.
```

Viewpoint of File_2.txt:

1c1	line 1
<	removed
>	added

Find lines with search expression - *grep*

Display lines from files matching some expression

```
grep PATTERN FILENAME
```

Example

```
grep file *.txt
```

```
File_1.txt:This is the first file!
File_2.txt:This is the second file.
```

Some options:

`-i` ignore case

`-l` just list name of files with matches

`-v` invert match (show non-matching lines)

`-n` print also line number of match

Wildcards

Before, we used

```
grep file *.txt
```

to select all files ending with `.txt`

`*` is wildcard for ‘zero or more characters’

(wildcard is expanded by the shell, not the application ‘grep’)

Possible – more specific – would be

```
grep file FILE_?.txt
```

to select all files where at ‘`?`’ one character or number stands

`?` is wildcard for ‘any single character’

(choosing, e.g. multiple characters, e.g. 2 with `??` is also possible)

Redirections - >, >>

For Unix and Linux

- the screen is “standard output”
- the keyboard is “standard input”

Standard output can be re-directed into a file

e.g. `cat File_1.txt > File_3.txt`

(Now File_3.txt is identical to File_1.txt)

More

`cat File_2.txt > File_3.txt`	Overwrite File_3.txt
`cat File_2.txt >> File_3.txt`	Add content of File_2.txt to File_3.txt
`cat > File_4.txt`	Read input from keyboard (simple editor – end with ctrl-C)

Pipe - |

Suppose you do a ‘grep’ and get a very long list.

Ways to handle this:

1. Redirect output into a file (> FILE) – creates an unnecessary file
2. Use a “pipe” to give output to “less”:

```
grep is File*.txt | less
```

Pipe | connects standard output of “grep” with standard input of “less”

More fancy

```
grep is File*.txt | sort | less
```

“sort” performs alphabetic sorting, “less” displays the result

Compressing and archiving

Compress a large file by

```
gzip FILENAME
```

This creates a compressed file FILENAME.gz

For many files or directories, one can create an archive, e.g.

```
tar -cvf FILES.tar FILE*.txt
```

tar = “tape archive”

Luckily, it also can create files

The arguments are important:

`-c` create

`-v` verbose (shows what is put into the tar file)

`-f` file

`-z` add compression

`-x` unpack (later)

`-t` list contents of .tar archive (use with –f and –v)

Alternative archiving/compressing

Compress and archive files by

```
zip ARCHIVE_NAME FILE1 FILE2 …
```

This creates a compressed file ARCHIVE_NAME

Unpack with

```
unzip ARCHIVE_NAME
```

Properties

- `zip` is compatible with zip on Windows
- `zip` always compresses (in contrast to ‘tar’)

vi – a text-based standard editor

There is an editor on any Unix/Linux system: `vi`

- One might occasional find oneself in it (because of $EDITOR)
- It's a very powerful editor
- It's very convenient to do small changes (don't need to graphical editor)
- When started it's in command mode (handle commands not write text)

Some important commands:

`i`	insert text left from curser
`a`	insert text right from curser or at end of line
`o`	insert new line below curser
`<ESC>`	return to command mode
`:w`	write (or `:w filename` to write with a new name)
`:q`	exit vi (`q!` to exit without saving when text was changed)
`:wq`	write and exit

Retrieving former commands

Imagine you did a really complicated command

```
myprogram ARG1 ARG2 ARG3 … ARG51
```

And want to repeat it. You don't need to retype!

1. If you want to repeat a very recent command

Select with curser keys (up/down)

2. Start writing and use Tabulator key to complete

3. Other way – if you want to repeat the last execution of a certain command, e.g

```
!my
```

Repeats the last command that *started* with 'my'

4. Further possibility:
   1. Type `history` (This prints a list of formerly executed commands)
   2. Search for number of command to be repeated
   3. Execute `!NUMBER`

Remote connections

To access a remote computer (e.g. your neighbor's computer), you have to log in

Connection with 'Secure Shell':

`ssh USERNAME@COMPUTERNAME` (`ssh myid@blogin.hlrn.de`)

or `ssh USERNAME@IP-Address` (`ssh myid@130.73.233.1`)

To connect, you will be asked for you password.

Useful option:

`-X` enable graphics forwarding ('X11')

Copying files remotely

Copy files from one compute to a different one.

Use ‘Secure Copy’:

```
scp myfile USERNAME@COMPUTERNAME:DIRECTORY
```

e.g.

```
scp File_1.txt myid@blogin.hlrn.de:/gfs1/work/myid
```

Useful options:

`-r` recursive copying (to copy whole directories)

`-p` preserve file properties (e.g. creation time)

Environment variables

Operating system stores environment variables for every process

In the standard shell (“bash”), list with ‘export’ (sometimes ‘set’)

An excerpt of ‘export’:

```
declare -x CPU="x86_64”
declare -x GPG_TTY="/dev/pts/0”
declare -x HISTSIZE="1000"
declare -x HOME="/home/hzkurs50"
declare -x HOST="linux-24ib”
declare -x LC_CTYPE="en_US.UTF-8"
declare -x LD_LIBRARY_PATH="/usr/lib64/mpi/gcc/openmpi/lib64”
declare -x LOGNAME="hzkurs50”
declare -x
PATH="/usr/lib64/mpi/gcc/openmpi/bin:/home/hzkurs50/bin:/usr/local/bin:/usr/bi
n:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games"
declare -rx PROFILEREAD="true"
declare -x PWD="/home/hzkurs50”
declare -x SHELL="/bin/bash”
```

Accessing and changing env. variables

Access environment variable with $ in from of name, eg. `$PATH`

Print value of variable

```
echo $PATH
```

Set new exported variable

```
export MYVAR=value
```

One can also extend an existing variable

e.g.

```
export PATH=$PATH:~/bin
```

(in $PATH the execution directories are listed with separator ':')

$PATH

$PATH defines the shell's search path for programs

```
export PATH=$PATH:~/bin        (1)
```

and

```
export PATH=~/bin:$PATH    (2)
```

are different!

$PATH is searched from left to right

- If you put a program '`ls`' in ~/bin and execute `ls`
  - (1) Executes `ls` in `/usr/bin`
  - (2) Executes `ls` in `~/bin`

- If a program is in a directory in $PATH:
  run with `PROGNAME`
- If the program directory is not in $PATH
  run with `./PROGNAME`

Configuring the shell

To make working with a shell more convenient and efficient

- edit configuration file

  `.bashrc`

e.g. add your favorite aliases:

```
alias sl="ls"
alias bhlrn="ssh myid@blogin.hlrn.de"
alias h="history 30"
alias a="alias"
alias adios="exit"
alias ciao="exit"
```

Or define a new prompt like

```
PS1="\h:\w[\#]$ "
```

File ownership and permissions

```
-rwxr-----  1 hbkurs50 workshop  134 Dec 13 11:16 example.sh
```

Each file belongs to a user and a group:

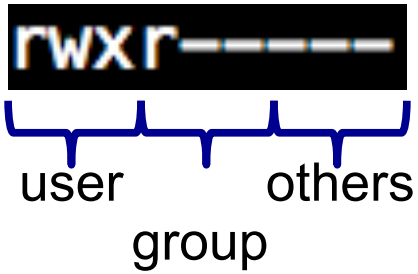
user: hbkurs50

group: workshop

Each file has access permissions:

`r` – read `w` – write `x` – execute

Displayed are 3 groups:



Here:

- User can read, write, and execute
- Group can read
- Others have no access

(For directory, execute means `cd` into it is allowed)

Changing ownership and permissions

Change owner:

```
chown USER.GROUP FILENAME
```

e.g.

```
chown hbkurs49.workshop FILE_1.txt
```

Change permissions with

```
chmod
```

e.g.

`chmod g+x DIRNAME`	Allow group members to cd into DIRNAME
`chmod u+x FILENAME`	Allow yourself to execute some file
`chmod go-rwx .`	Remove any permissions for current directory for group and others

Shell scripts – the full power of shells

- Create your own shell script for repeating operations
- e.g. for administration, when we add a new user we create 2 directories and set permissions with

```
./newuserdirs.sh USER GROUP
```

- The script is

```
#!/usr/bin/sh
echo USAGE:
echo Call as newuserdirs.sh USER GROUP

echo Creating directories for user: $1

mkdir /iblade/home/$1
mkdir /iblade/user/$1
chown ${1}.${2} /iblade/home/$1
chown ${1}.${2} /iblade/user/$1
```

Programming features in shell scripts

Shells support programming features

- Variables (including arrays)
- Loops
- If-clauses
- functions

Loop Example: Rename a set of numbered files

```
#!/bin/bash

for i in 1 2 3 4 5 6
do
    mv File_${i}.txt Example_${i}.txt
done
```

Summary - commands

Shells are useful for various file handling operations

Important commands

`cd`	Change directory
`ls`	List directory content
`mkdir / rmdir`	Make and remove directories
`rm`	Remove files
`mv / cp`	Copy and move/rename files or directories
`man`	Display manual for some command
`cat / more / less`	Display file contents
`diff`	Show differenced in files
`grep`	Search in a file
`tar/zgip/zip/unzip`	Create archives and compress files
`sort`	Sort shell outputs
`history`	Display history of executed shell commands
`ssh`	Secure Shell remote connection
`scp`	Copy files between computers
`vi`	Classic powerful standard editor

Summary - specialties

Some characters and combinations with special meaning

`.`	The current directory
`..`	The parent directory
`~`	The home directory
`>`	Redirection into a file
`>>`	redirect and append to a file
`<`	Redirected input
`|`	Pipe – directs standard output of one command to standard input of a second command
`/`	The root directory and directory name separator
`$`	Marks an environment variable
`&`	Following a command: Start in the background
`*`	Wildcard for zero or several characters
`?`	Wildcard for exactly one character
`!NUMBER`	Execute command with number ‘NUMBER’
`!MY`	Execute last command that started with ‘MY’

Some exercises

1. List the files in /tmp according to their size order from small to large
2. List the root directory in long format showing the time of last access instead of creation time.
3. How do you tell diff to ignore changes that just insert or delete blank lines?
4. Create a file exercise4.txt and fill it with some text. What happens when you do ‘cat exercise4.txt > exercise4.txt’
5. What does the command ‘wc’ do?
6. Create a file exercise6.txt and fill it with size lines of different numbers. How do you use ‘cat’ and ‘sort’ to display the numbers ordered from large to small?
7. How can one delete all files whose names start with ‘b’, have length of 4 plus the file ending .txt.
8. What is the difference between `cd HOME` and `cd $HOME`?
9. How can you list all files in a directory that contain that word ‘workshop’?
10. What is the purpose of the command ‘head’?
11. How can you display the last 10 lines of a file?
12. How do you set the permissions of a script so that only member of your group may execute it?