Betriebssysteme

Entwicklungsmethodik

Literate Programming

Slide set 2:

Crash courses C / bash

for self-study

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Winter semester 2015/16

Dr. Hans-Georg Eßer

h.g.esser@gmx.de

http://ohm.hgesser.de/

v1.0, 10/06/2013

Self-study

- I have these slides in the course "System programming Unix / Linux" in Used in the 2013 summer semester
- The videos can be found at

  http://ohm.hgesser.de/sp-ss2013/

Introduction to C (1)

- First of all the most important:
  - no classes / objects
    - → instead of objects:
      "Structs" (composite data types)
    - → instead of methods just functions
    - → Always pass the variables to be processed as arguments
  - no string data type (but character arrays)
  - frequent use of pointers
  - int main () {} is always the main program

Introduction to C (2)

- More detailed information for
  Self study: http://www.c-howto.de/

  → on the website: more detailed version with explanatory comments
  (Download: Zip archive)

- also available as a book for approx. 20 €

- In the lecture / exercise: focus on
  differences to C ++ / C # / Java

Introduction to C (3)

- Following this lecture: first exercise sheet with C tasks
- Some preparatory information too
  - **Structs (** Structures, compound types)
  - **Pointers**

Introduction to C (4)

Structs

- Several ways of declaration

```
struct {
    int i;
    char c;
    float f;
} variable;

variable.i = 9;
variable.c = 'a';
variable.f = 0.123;
```

```
struct mystruct {
    int i;
    char c;
    float f;
};

struct mystruct variable;

variable.i = 9;
variable.c = 'a';
variable.f = 0.123;
```

```
typedef struct {
    int i;
    char c;
    float f;
} mystruct;

mystruct variable;

variable.i = 9;
variable.c = 'a';
variable.f = 0.123;
```

Introduction to C (5)

pointer

- Declaration with *: `char * ch_ptr;`
- manage memory addresses (where is the variable located?)
- Operators
  - & (Address from)
  - * (Dereferencing)

```
char ch, ch2;
char * ch_ptr;          char * ch_ptr2;

ch_ptr = & ch;          // address of ch? // content
ch2 = * ch_ptr;

ch_ptr2 = ch_p          tr;
      // copied         only address
```

Introduction to C (6)

- Struct and pointer combined
- Often with linked lists

```
struct list {
    struct list * next;
    struct list * prev;
    int content;
};

struct list * start;
struct list * p;

for (p = beginning; p! = NULL; p = p-> next) {}
    use (p-> content);
```

More about C

- Program and header files
  - Header files (* .h) contain function prototypes and macro definitions (but not normal code)
  - Program files (* .c) contain the code, but can also contain prototypes and macros (no requirement to create a .h file)

More about C

- Functional prototypes
  - allow the use of functions, the implementation of which is further down in the program (or in another file)
  - Prototype only contains return type, name and arguments, e.g. B.

    int sum (int x, int y);

More about C

- How does the compiler find the header files?

  → Two options:

  - # include "path / to / file.h"

    Filename is path (relative to the directory with the . c file)

  - # include <name.h>

    name.h is searched for in the standard include directories.

    Which are they? Set when building the gcc ...

More about C

- Standard include directories

```
[ esser @ s15337257 : ~] $ cpp -v
Using built-in specs.
Target: i486-linux-gnu
[...]
# include "..." search starts here:
# include <...> search starts here: / usr / local / include

  /usr/lib/gcc/i486-linux-gnu/4.4.5/include
  /usr/lib/gcc/i486-linux-gnu/4.4.5/include-fixed
  / usr / include
End of search list.
```

(and their sub-folders)

pointer

- Pointer types
  - type * ptr;
    → ptr is a pointer to something of type Type
  - type ** pptr;
    → pptr is a pointer to a pointer of the type Type
  - ptr or. pptr are memory addresses
  - * ptr returns the value stored in the memory location to which ptr show
  - analogous: ** pptr is a value, but * pptr a pointer

pointer

- Pointer types
  - & Operator creates a pointer for the variable
  - Examples:

```
int i;
int * ip;
int ** ipp;

i = 42;
ip =?                          // ip = address of i
ipp =?                           // ipp = address of ip

printf (* ip); // -> 42
printf (** ipp); // -> also 42
```

pointer

- Uninitialized pointers: bad
  - Example:

```
int * ip;
int ** ipp;

printf (ip);                    // not init. Address (0) // also illegal, write to
printf (* ip); // illegal -> abort

* ip = 42;
                                // not def. address
```

pointer

- Be careful with char * a, b, c; Etc.

```
[ esser @ macbookpro : tmp] $ cat t2.c
int main () {
    char * a, b;
    printf ("| a | =% d \ n", sizeof (a)); printf ("| b | =% d \ n",
    sizeof (b));
}

[ esser @ macbookpro : tmp] $ gcc t2.c; ./a.out
| a | = 8
| b | = 1
```

- better: char * a, * b, * c;

Introduction to Bash

- Numerous shells (command line interpreters) are available for Unix / Linux
  (C-Shell csh, Korn Shell ksh, Bash, tcsh)
- Linux standard shell:
  Bash ("Bourne Again Shell")
- compared to Windows shells:
  - significantly more powerful than CMD.EXE (Command.com)
  - very different to use than PowerShell

Shell prompt (1)

- Shell shows through **prompt** indicates that she is ready to take an order
- Prompts can look different:
  - ... $ _

    ...> _: User prompt, not privileged
  - … # _: Root prompt for the administrator

Shell prompt (2)

- Before the $,>, # mostly references to user, computer, working directory

  [esser @ macbookpro : SysPro] $

  root @ quad : ~ #

  - esser, root: User name; individually
  - macbookpro, quad: Computer name
  - SysPro, ~: Working directory, also in full length depending on the prompt setting (e.g.
    / home / esser / data / Ohm / SS2012 / SysPro)
  - ~ = "Home directory" of the user

Command input (1)

- Enter the command at the prompt and send it with [Enter]

- Shell tries (usually) to interpret the first word as a command name:
  - Alias? (→ later)
  - Shell internal function? (→ later)
  - built-in shell command? (e.g. CD)
  - external program? (Search in path)

Command input (2)

- Example: News **Working directory**
  Show (pwd = **p** rint **w** orking **d** irectory)

```
[ esser @ quad : ~] $ pwd
/ home / esser
[ esser @ quad : ~] $ _
```

- After processing the command (often: with an "answer") the prompt appears again - the shell is ready for the next command

Command input (3)

- Send several commands at once: with semicolon; separate from each other

```
[ esser @ quad : ~] $ pwd; pwd
/ home / esser
/ home / esser
[ esser @ quad : ~] $ _
```

Command input (4)

- **Table of Contents** Show: ls (**l** i **s** t)
- always refers to the current working directory (alternative: specify location as parameter)

```
[ esser @ quad : ~] $ ls
bahn-2011-02-22.pdf                bh-win-04-kret.pdf
book_kap08.pdf                     bv-instructions.pdf
bz2.pdf
[ esser @ quad : ~] $ ls / tmp
cvcd kde-esser ksocket-esser orbit-esser ssh-vrUNLb1418 virt_1111

[ esser @ quad : ~] $ _
```

Command input (5)

- Content with more information: ls -l

```
[ esser @ quad : ~] $ ls -l
- rw ------- 1 esser users                29525 Feb 21          2011 bahn-2011-02-22.pdf
- rw-r - r-- 1 esser users               745520 Apr 10          2004 bh-win-04-kret.pdf
- rw-r - r-- 1 esser users               856657 Oct 21          2005 book_kap08.pdf
- rw-r - r-- 1 esser esser               738570 Mar 17        20:29 bv-instructions.pdf
- rw-r - r-- 1 esser users [ esser @ quad :  123032 Sep 22          2003 bz2.pdf
 ~] $ _
```

- Edition also contains:
  - Access rights (- rw-r - r-- Etc.) → later
  - File owner and group (esser, users) → later
  - Size and date / time of the last change

Command input (6)

- Create empty file (for experiments): touch

  [esser @ quad : ~] $ **touch test file**
  [esser @ quad : ~] $ **ls -l test file**
  - rw-r - r-- 1 esser esser 0 Apr 7 13:58 test file [esser @ quad : ~] $ _

- File is size 0

Command input (7)

- Error messages: Unknown command

  [esser @ quad : ~] $ **fom**
  No command 'fom' found, did you mean: Command 'fim' from package 'fim'
  (universe) Command 'gom' from package 'gom' (universe) Command 'fop'
  from package 'fop' (universe) Command 'fdm 'from package' fdm
  '(universe) Command' fpm 'from package' fpm2 '(universe) [...]

  fom: command not found [esser @
  quad : ~] $ _

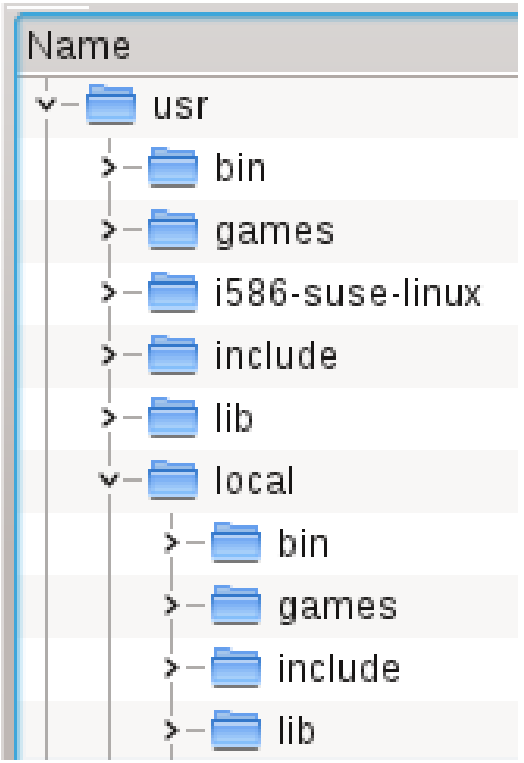
- The message can also be in German

File Management (1)

Basics (1)

- Linux knows no "drive letters" (C :, D: Etc.)

- Root directory is called /
- Path separator: also / - ie:
  / usr / local / bin is this
  directory am in the register
  local in the register usr.
  (as with web addresses)



File Management (2)

Basics (2)

- Additional data carriers appear in sub-folders
  - Example: DVD with files has volume name SYSPRO
  - file test.txt is at the top level of the DVD directory
    when / media / SYSPRO / test.txt reachable
    (Windows: e: \ test.txt)
  - file Software / index.html the DVD accordingly
    when / media / SYSPRO / Software / index.html
    (Windows: e: \ Software \ index.html)

File Management (3)

Basics (3)

- Each user has their own for private user data **Home directory,** that is usually below / home lies, e.g. B. / home / esser.
- The tilde ~ is always an abbreviation for the home directory
  - also works in compound paths
  - ~ / Data / letter.txt instead of /home/esser/Daten/brief.txt

File Management (4)

Basics (4)

- Exception: the home directory of the system administrator root is not
  / home / root, rather / root
- The trick with the tilde ~ also works
  For root
- Why? / home could be on a separate partition and not be available in the event of a false start

File Management (5)

Basics (5)

- Two special directories in each folder
  - . . if the directory is one level lower (from / usr / local / bin it's off .. so / usr / local)
  - . is the current directory
- Paths can be **absolutely** and **relative** build together
  - absolute path begins with /
  - relative path not; it always applies from the current working directory

File Management (6)

Directory navigation

- command cd (**c** hang **d** irectory) changes to another directory

- Target directory as argument of CD specify - optionally with relative or absolute path

[esser @ quad : ~] $ **pwd**
/ home / esser
[esser @ quad : ~] $ **cd / home; pwd**
/ home
[esser @ quad : home] $ **cd ..; pwd**
/
[esser @ quad : /] $ _

File Management (7)

Copy file

- command cp (**c** O **p** y) copies a file
- Sequence: cp *Original copy*

[esser @ quad : tmp] $ **ls -l**
- rw-r - r-- 1 esser wheel 1501 Apr 5 11:37 test.dat
[esser @ quad : tmp] $ **cp test.dat copy.dat**
[esser @ quad : tmp] $ **ls -l**
- rw-r - r-- 1 esser wheel 1501 Apr 8 12:17 pm copy date
- rw-r - r-- 1 esser wheel 1501 Apr [esser @ quad : tmp] $ _ 5 11:37 test.dat

! Copy is given the current date / time

File Management (8)

Rename file

- command mv (**m** O **v** e) renames a file
- Sequence: mv *OldName NewName*

[esser @ quad : tmp] $ **ls -l**
- rw-r - r-- 1 esser wheel 1501 Apr 5 11:37 test.dat [esser @ quad
: tmp] $ **mv test.dat new.dat**
[esser @ quad : tmp] $ **ls -l**
- rw-r - r-- 1 esser wheel 1501 Apr 5 11:37 new.dat [esser @ quad : tmp] $ _

! Renaming does not change the date / time

File Management (9)

Move file

- command mv (move) moves a file
- Sequence: mv *Oldname newfolder /*

```
[ esser @ quad : tmp] $ ls -l
- rw-r - r-- 1 esser wheel 1501 Apr 5 11:37 test.dat
[ esser @ quad : tmp] $ mv test.dat / home / esser /
[ esser @ quad : tmp] $ ls -l
[ esser @ quad      : tmp] $ ls -l / home / esser /
- rw-r - r-- 1 esser wheel 1501 Apr                    5 11:37 test.dat
       [...]
[ esser @ quad : tmp] $ _
```

! Moving does not change the date / time

File Management (10)

delete file

- command rm (**r** e **m** ove) deletes a file

[esser @ quad : tmp] $ **ls -l**
- rw-r - r-- 1 esser wheel 1501 Apr 5 11:37 test.dat [esser @ quad : tmp] $ **rm test.dat**

[esser @ quad : tmp] $ **ls -l**
[esser @ quad : tmp] $ _

File Management (11)

Multiple files

- Some commands take multiple arguments; B.
  - mv (when moving to another folder)
  - rm
- Examples:

[esser @ quad : tmp] $ **mv file1.txt file2.txt folder /**
[esser @ quad : tmp] $ **rm file3.txt file4.txt file5.txt**
[esser @ quad : tmp] $ _

File Management (12)

Wildcards (*,?)

- You can also use wildcards for commands that accept multiple arguments:
  - * stands for any number of characters (including 0)
  - ? stands for exactly one character
- Examples:

[esser @ quad : ~] $ **ls -l ?????. pdf**
- rw-r - r-- 1 esser staff 79737 Apr 2 01:18 RegA4.pdf
- rw-r - r-- 1 esser staff 132246 Apr 4 18:02 paper.pdf [esser @ quad : ~] $ **rm / tmp / ***

[esser @ quad : ~] $ _

Test commands

- Delete command with wildcards too daring?

  → beforehand with echo testing:

```
[ esser @ quad : Downloads] $ echo rm * .zip
rm Logo_a5_tif.zip Exercise1.zip c32dwenu.zip ct.90.01.200-209.zip
ct.90.12.130-141.zip
ct.91.02.285-293.zip ct.91.12.024-025-1.zip
ct.91.12.024-025.zip ct.92.08.052-061.zip
ix.94.03.010-011.zip ix.94.07.068-071.zip
[ esser @ quad : Downloads] $ rm * .zip
[ esser @ quad : Downloads] $ _
```

Wildcard resolution

- The last example reveals something about resolving the wildcards
  - If you rm * .zip the shell starts ***Not*** rm with the argument "*. zip "
  - The shell searches for all suitable files in the current directory and makes each file name an argument for the rm- Call.
  - E. sw L. ir O d G ao
    rm O l_s a5_tif.zip Exercise1.zip c32dwenu.zip ct.90.01.200-209.zip ... called.

Directories (1)

You can do similar things with directories as you can with files

- Create Directory
- Delete (empty!) directory
- Rename or move directory
- Delete directory recursively (with all files and subfolders it contains)

Directories (2)

Create Directory

- command mkdir (**m** a **k** e **to you** ectory) creates a new (empty) subdirectory

```
[ esser @ quad : tmp] $ ls -l
[ esser @ quad : tmp] $ mkdir under
[ esser @ quad : tmp] $ ls -l
drwxr-xr-x 2 esser wheel 68 Apr [ esser @ quad                8 14:28 under
                                : tmp] $ cd under
[ esser @ quad : under] $ ls -l
[ esser @ quad : under] $ cd ..
[ esser @ quad : tmp] $ _
```

! short form md For mkdir not always present → avoid

Directories (3)

Delete directory

- command rmdir (**r** e **m** ove **to you** ectory) deletes an empty (!) subdirectory

[esser @ quad : tmp] $ **touch under / file**
[esser @ quad : tmp] $ **rmdir under**
rmdir: under: directory not empty [esser @ quad : tmp] $ **rm under / file**
[esser @ quad : tmp] $ **rmdir under**
[esser @ quad : tmp] $ _

! short form approx For rmdir not always present → avoid

Directories (4)

Rename / move directory

- works like renaming / moving files

- same command: mv, again two variants:
  - mv directory new name
  - mv directory other folder /

Directories (5)

Delete directory recursively

- command rm (**r** e **m** ove) has an option **-** r

  to the **r** eursive deletion:

[esser @ quad : tmp] $ **mkdir a; mkdir a / b; mkdir a / b / c**
[esser @ quad : tmp] $ **touch a / b / c / file**
[esser @ quad : tmp] $ **rmdir a**
rmdir: a: Directory not empty [esser @ quad : tmp] $ **rm**
-ra
[esser @ quad : tmp] $ _

! Be careful with recursive deletion: "What is gone is gone"

~~Undelete~~

- Undelete = Undelete
  - does not exist under Linux
  - Recovery of deleted files with ProfiTools possible if the computer was switched off immediately after deletion
  - however, such tools restore a large number of files → enormous effort to then find the file you are looking for; among other things, the file names are permanently lost

→ in front rm -r ... check several times ...

Options and arguments

- **Arguments:** z. B. filenames; often refer to objects that are to be manipulated
- **Options:** change the behavior of a command

  - there are two variants for most commands:
  - short options: - a, -b, -c, ...

    → can be combined: - abc = -a -b -c
  - long options: - ignore, --force, --all Etc.
  - Example: - r at rm

Help: manual

- Most commands have a so-called man page, the

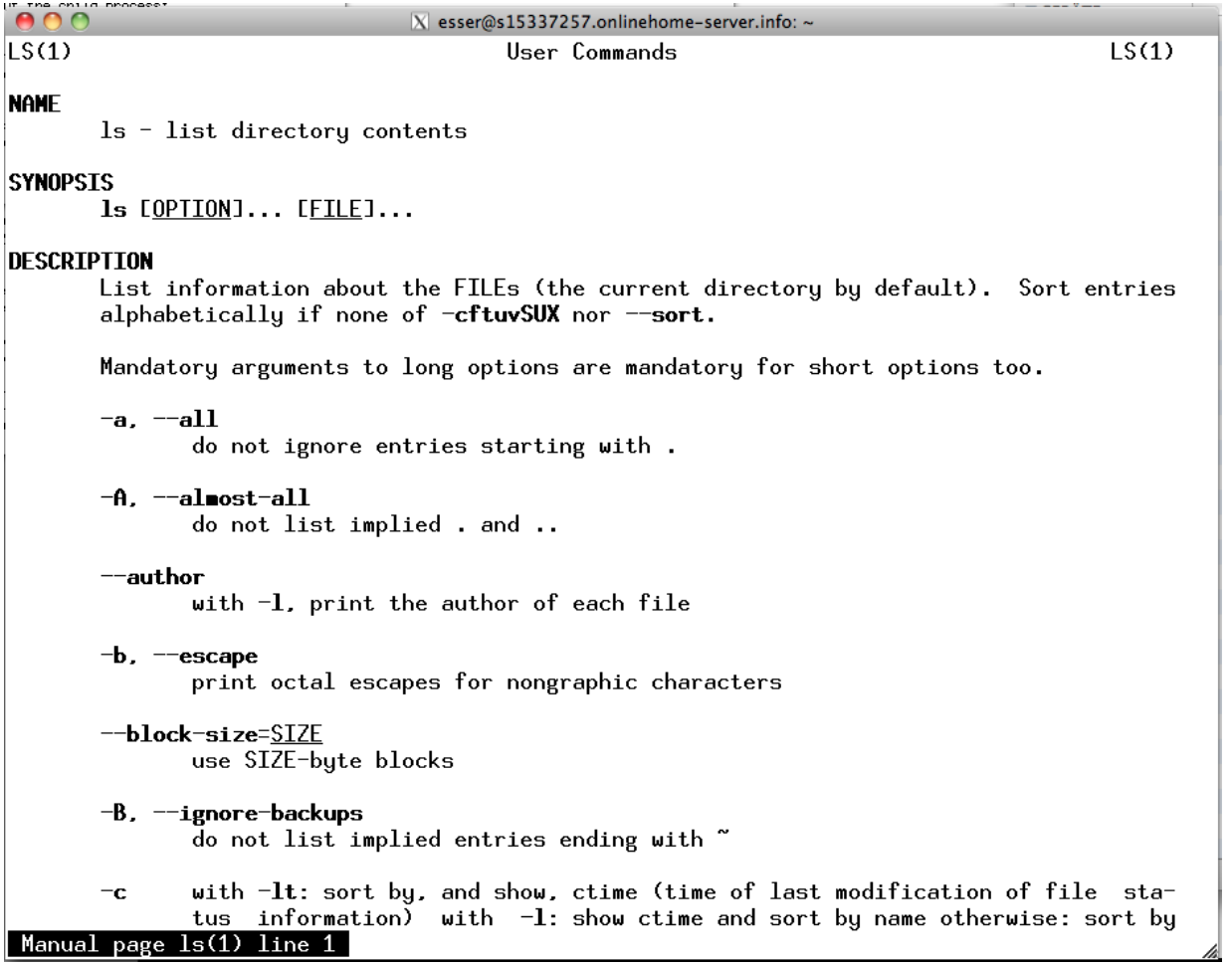
  You over

  man command

  recall

- Example:

  man ls



The vi editor (1)

- Standard editor on all Unix systems (and thus also Linux): vi (**vi** sual editor)
- Operation takes getting used to
- two modes of operation
  - **Command mode (** activated after start; Normal mode)
  - **Edit mode**
- vi started accidentally? Exit without saving changes with [Esc]: q!

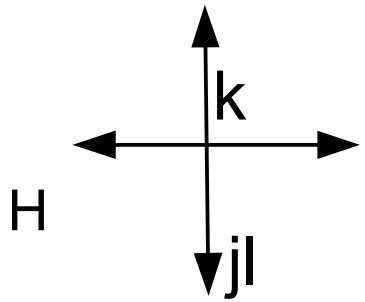
The editor vi (2)

- Why dealing with vi learn?
  - on every - however minimalist - Unix system is a vi installed (small program):
    [esser @ quad : ~] $ **ls -l / usr / bin / vi / usr / bin / emacs**
    - rwxr-xr-x 1 root root 5502096 Nov 9 2008 / usr / bin / emacs
    - rwxr-xr-x 1 root root 630340 Oct 17 2008 / usr / bin / vi
  - runs in the terminal → helpful for remote access
  - In the event of problems (disk errors, not all file systems available), other editors may not be available,
    vi maybe yes → unfortunately no longer applies to current Linux versions

The editor vi (3)

- Switch to edit mode: i, I, a, A
  - i: Insert text in front of the cursor
  - a: Insert text after the cursor
  - I: Insert text at the beginning of the line
  - A: Insert text at the end of the line
- Exit edit mode: [Esc]
- Navigating in the text:

  Cursor keys or:



The editor vi (4)

- Delete characters / text:
  - in edit mode with [Backspace] and [Del], as known from other editors
  - several options in command mode:
  - x deletes characters under cursor
  - X deletes characters to the left of the cursor
  - dw deletes from cursor position to the beginning of the next word
  - dd deletes the current line
  - number in advance: multiple execution (15dd: 15 lines)

The editor vi (5)

- Save and exit
  - Always in command mode first
    → if in doubt, press [Esc] once
  - To save: : w
  - Save (force):: w!
  - Exit (only works if the text has not been changed since it was last saved):: q
  - Force exit (without saving):: q!
  - Save and exit: : wq (or: ZZ without ":")

The editor vi (6)

- Search in the text
  - Forward search: / and search term, then [Enter]
  - Jump to the next hit: n (next)
  - Reverse search:? and search term, then [input]
  - Jump to the next hit: n
  - Switching between forward and backward search: simple / or? , then enter and with n keep looking (in a new direction)

The editor vi (7)

- Undo / redo
  - Undo last change: u (undo)
  - also works several times: u, u, u, ...
  - ... and with multiple options: 3u undoes the last three changes
  - Cancel an undo step: [Ctrl] + r: redo
  - multiple redo: e.g. B. 3 [Ctrl] + r

The vi editor (8)

- Copy & Paste: Copy ...
  - yw (from cursor position to end of word)
  - y $ (from cursor position to end of line)
  - yy (whole line)
  - 3yy (three lines from the current one)
- ... and paste
  - P (inserts the contents of the buffer at the cursor position)
- Cut & Paste
  - Delete with dd, dw Etc.; then paste with P

The editor vi (9)

- Copy & paste with the mouse
  - If you use the graphical user interface, you can also use the mouse:
  - Copy: Mouse pointer on 1st character, click (and hold), drag to the last character, release
  - Insert: move the cursor to the target, then (in insert mode!) Press the middle mouse button
  - For both steps you have to depending on vi Version possibly press the [Shift] key

The editor vi (10)

- Open file in editor:

  [esser @ quad : ~] **vi filename**

- load second file at cursor position:

  : read filename

  (in command mode!)

Shell variables (1)

- Use the shell (and other programs too) **Environment variables (** for options, settings etc.)
- "Set" outputs a list of all variables set in this shell

```
$ set
BASH = / bin / bash
BASH_VERSION = '3.2.48 (1) -release'
COLUMNS = 156
COMMAND_MODE = unix2003
DIRSTACK = ()
DISPLAY = / tmp / launch-Lujw2L / org.x: 0
EUID = 501
GROUPS = ()
HISTFILE = / home / esser / .bash_history
HISTFILESIZE = 500
HISTSIZE = 500
HOME = / home / esser
HOSTNAME = macbookpro.fritz.box
. . .
```

Shell variables (2)

- You output individual variables with “echo” and a dollar sign ($) in front of the variable name

```
$ echo $ SHELL
/ bin / bash
$ _
```

- to change / set write "var = value":

```
$ TESTVAR = form
$ echo $ TESTVAR
fom
$ set | grep TEST
TESTVAR = form
$ _
```

- You can use variables too **export:**

```
$ export TESTVAR
$ _
```

→ next slide

Shell Variables (3)

- Export?

  The value of a variable is only valid locally in the running shell.

- Exported variables also apply in programs started from the shell

```
$ A = one; B = two; export A
$ echo "A = $ AB = $ B"
A = one B = two
$ bash                          #   start new shell; this is a new program!
! $ A = one B =
  echo "A = $ AB = $ B"
  $ exit                        #   exit this second shell, back to the first
  $ echo "A = $ AB = $ B"
  A = one B = two
```

Shell Variables (4)

- "Export" outputs a list of all exported variables without an argument - but in an unusual syntax

```
$ export
declare -x A = "1"
declare -x Apple_PubSub_Socket_Render = "/ tmp / launch-CYfDhh / Render" declare -x COMMAND_MODE =
"unix2003"
declare -x DISPLAY = "/ tmp / launch-Lujw2L / org.x: 0"
declare -x HOME = "/ Users / esser"
declare -x INFOPATH = "/ sw / share / info: / sw / info: / usr / share / info" declare -x LOGNAME =
"esser"
. . .
```

- (Background: "declare -x VAR" also exports the variable VAR, so it is the same as "export VAR")

History (1)

- Shell remembers the commands entered ("History")

- Complete output with "history":

```
$ History
1 df -h
2 ll
3 / opt / seamonkey / seamonkey
4 dmesg | tail
5 ping hgesser.de
6 google-chrome
7 killall kded4
```

- How many entries? Normal 500:

```
$ echo $ HISTSIZE
500
```

History (2)

- In addition to the output of the complete history, there is also an intelligent search for old commands: [Ctrl-R]

```
$ # Look for the last echo call
$ ^ R.
(reverse-i-search) ` ech ': echo $ HISTFILESIZE
```

- execute with [Enter]
- further [Ctrl-R] return older hits
- Also: Scroll through old commands with [arrow up], [arrow down]
- found command can be accepted and revised

Filters for text streams

- Idea for the filter:
  - Convert standard input to standard output
  - Assemble chains of filters:
  - prog1 | filter1 | filter2 | filter3 ...
  - with input file:
    prog1 <input | filter1 | ...
- cat, cut, expand, fmt, head , od, join, nl, paste, pr, sed, sort, split, tail , tr, unexpand, uniq, WC

cat

- cat stands for con **cat** enate (join)
- outputs several files in quick succession

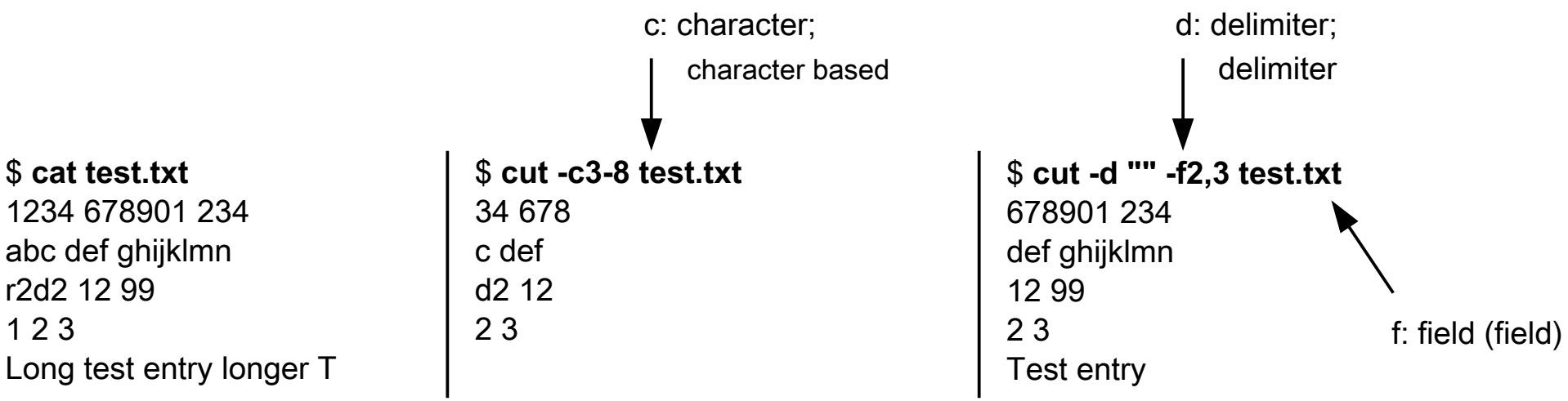
- only one file if desired

  → Mini file viewer

- Special options:
  - - n (line numbers)
  - - T (show tabs as ^ I)
  - ... and some more (see: man cat)

cut

- cut can cut text column by column - columns can optionally be defined via
  - Character positions
  - Separator (which separate logical columns from one another)



fmt

- fmt (**f** or **m** a **t)** breaks text files

no Upheavals

$ **cat test.txt**
This is an example of a sentence. That's an example for a sentence. This is an example of a sentence. That's once

n example of a sentence. This is an example of a sentence. n example of a sentence. This is an example of a
This is an example of a sentence. That's an example for a sentence. This is an example of a sentence. That's once

sentence. This is an example of a sentence. This is an example of a sentence. This is an example of a sentence. This

$ **fmt test.txt**
is an example of a sentence.
for a sentence. This is an example of a sentence. This is an example of a sentence. This is an example for

an example of a sentence. This is an example of a sentence. This is an example of a sentence. This is once a

Line breaks

- Parameter -w75: width 75 (width)

split

- split can split large files into multiple files with a specified maximum size
- (cat then reassembles them)

```
$ split ZM_ePaper_18_11.pdf -b1440k ZM_ePaper_18_11.pdf.
$ ls -l ZM *
- rw-r - r-- 1 esser esser 10551293 2011-04-29 06:58 ZM_ePaper_18_11.pdf
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.aa
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ab
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ac
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ad
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ae
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.af
- rw-r - r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ag
- rw-r - r-- 1 esser esser                    229373 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ah
$ cat ZM_ePaper_18_11.pdf. *> ZM_Kopie.pdf
$ ls -l ZM_Kopie.pdf
- rw-r - r-- 1 esser esser 10551293 2011-04-29 14:48 ZM_Kopie.pdf $ diff ZM_ePaper_18_11.pdf
ZM_Kopie.pdf

$ _
```

sort

- sort is a complex sorting tool that
  - Sort by ***n-*** the column
  - alphabetical and numerical sorting

  supported

- Simple examples:

$ **cat test3.txt**	$ **sort test3.txt**	$ **sort -n test3.txt**
13 cars	13 cars	avenue
5 trucks	2 trees	House
24 bikes	24 bikes	hotel
2 trees	5 trucks	Road
flat	avenue	flat
House	House	2 trees
hotel	hotel	5 trucks
Road	Road	13 cars
avenue	flat	24 bikes

uniq

- uniq (**uniq** ue, unique) combines several identical (consecutive) lines into one; so removes Doppler

- Alternative: When sorting with sort you can use the option -u (**u** nique) directly remove Doppler;

  - instead of sort file | uniq so better sort -u file

grep

- grep (**G** lobal / **r** egular **e** xpression / **p** rint) only shows the lines of a file that contain or do not contain a search term (option -v)

$ **wc -l / etc / passwd**
57 / etc / passwd
$ **grep esser / etc / passwd**
esser: x: 1000: 1000: Hans-Georg Esser ,,,: / home / esser: / bin / bash
$ **grep / bin / bash / etc / passwd**
root: x: 0: 0: root: / root: / bin / bash
esser: x: 1000: 1000: Hans-Georg Esser ,,,: / home / esser: / bin / bash
$ **grep -v / bin / bash / etc / passwd | head -n5**
daemon: x: 1: 1: daemon: / usr / sbin: / bin / sh
bin: x: 2: 2: bin: / bin: / bin / sh
sys: x: 3: 3: sys: / dev: / bin / sh
sync: x: 4: 65534: sync: / bin: / bin / sync
games: x: 5: 60: games: / usr / games: / bin / sh

sed (1/2)

- sed (**S.** tream **Ed** itor) performs (among other things) search / replace functions in a text

$ **cat test4.txt**
The word is a word and multiple words are the plural of the word. Without words or words there is no sentence - we are wordless.

$ **sed 's / word / picture /' test4.txt**
The image is a word and several words are the plural of image . Without words or image e there is no sentence - we are wordless.

$ **sed 's / word / OHM / G 'test4.txt**
The OHM is a OHM , and several words are the plural of OHM . Without words or OHM e there is no sentence - we are wordless.

$ **sed 's / word / OHM / g i 'test4.txt**
The OHM is a OHM , and several words are the plural of OHM . Without words or OHM e there is no sentence - we are OHM Come on.

s: substitute (**s /.../.../** gi)
g: global (s /.../.../ **G** i)
i: ignore case (s /.../.../ g **i)**

The i-option is not available in every sed version!

sed (2/2)

- sed options:
  - - i: in-place-editing, changes the specified file; ideally with a suffix for a backup file:

    z. B. sed -i.bak 's / word / image / g' test4.txt

    first creates a backup copy of test4.txt.bak and then

    changes test4.txt
  - - e: to combine multiple replacements; z. B.

    sed -e 's / 1 / one / g' -e 's / 2 / two / g' test.txt
  - More options → Manpage

Regular expressions

- Idea: More general search terms, comparable to wildcards (*,?) For file names
- Template:
  - . - any character
  - [abcd] - one of the characters a, b, c, d
  - [2-8] - one of the characters 2, 3, 4, 5, 6, 7, 8
  - ^ - start of line
- $ - end of line
- ? - The previous expression may appear, but does not have to be
- * - previous expression can occur any number of times (even 0 times)

$ **cat test5.txt**
House
The hotels
Hotels by the water
Construction house object
Not this line

$ **grep 'H. * s' test5.txt**
House
The hotels
Hotels by the water
Construction house object

$ **sed 's / H. * s / HAUS / g' test5.txt**
HOUSE
The HOUSE
Houses
Construction HOUSE object
Not this line