Old Software Treasuries

markus schnalke <meillo@marmaro.de>

this talk

- show the historical background
- introduce, explain and demonstrate `ed`, `sed`, `awk`
- solve real life problems

please interrupt me at any time in case of questions
please tell me what I should demonstrate

goal: motivate you to learn and use these programs

old software treasuries

- older than I am
- standardized
- various implementations
- freely available
- good software

origin

- classic Unix tools
- from Bell Labs
- representations of the Unix Philosophy

historical background

1968:

- MULTICS fails
- the last people working on it were Thompson, Ritchie, McIlroy, Ossanna

1969:

- they still think about operating systems
- pushing idea: a file system (mainly Thompson)
- in search for hardware: a rarely used PDP-7

historical background

1970:

- Kernighan suggests “Unix”
- deal: a PDP-11 for a document preparation system

1973:

- rewrite of Unix in C

1974:

- first publication

computer hardware back then

- computers are as large as closets
- users work on terminals
- terminals are: a keyboard and a line printer
- line printers have a speed of 10–15 chars/s

<picture of PDP-11>

impact on software

- don’t waste characters
- be quiet
- nothing like “screen-orientation”

ed

ed

- by Ken Thompson in early 70s, based on `qed`
- the Unix text editor
- the *standard* text editor!
- Unix was written with `ed`
- first implementation of Regular Expressions

ed: old-fashioned

- already old-fashioned in 1984
- “So why are we spending time on such a old-fashioned program?”
- “Although many readers will prefer some other editor for daily use, `ed` is universally available, efficient and effective.”

ed: reasons

- the standard (text editor)
- available as `/bin/ed`
- line oriented $\rightarrow$ works on any terminal
- needs few bandwidth
- good for automated editing (scripts)
- good for presentations (?)

it’s not so much about using it, but about knowing it

ed: overview

- usage: `ed [-s] [file]`
- commands: `[a1[,a2]] cmd [params]`
- default addresses, default params
- success → no feedback
- problems → “?”
- about 30 commands, half of them important

ed: commands

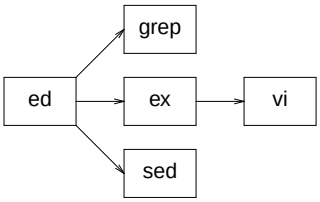
`i`	insert	`p`	print
`a`	append	`n`	print with line number
`c`	change	`l`	list characters
`d`	delete	`e`	edit file
`m`	move	`w`	write file
`t`	transfer (copy)	`q`	quit
`s`	substitute		
`u`	undo		
`g`	global command		

ed: addresses

`4`	4th line
`$`	last line
`.`	current line
`+`	next line (`.+1`)
`--`	the same as `.-2`
`/RE/`	next line matching RE
`?RE?`	previous line matching RE
`1,$`	all lines
`-,/foo/`	from previous line to next /foo/

<ed demo>

ed: derived software



sed

sed

- by Lee McMahon in 1973
- “stream editor”
- started as a one-night hack on `ed` (AFAIR)
- makes `ed` suitable for pipeline processing
- irony: today, people use `sed -i` for in-place editing

sed: differences to ed

- line processing cycle
- no forward/backward references
- hold space (`g`,`G`,`h`,`H`,`x`)
- labels and branches
- bad `i` and a syntax

<sed demo>

awk

awk

- by Aho, Weinberger, Kernighan
- `oawk` in 1977, `nawk` in 1985
- name is abbreviation of surnames (but also “awkward”)
- the second scripting language on Unix (besides `sh`)
- “ `sed` meets C”

awk: purpose

- for text processing
- avoids complex constructs in `sh`
- floating point arithmetic

awk: usage

- `awk 'commands' <in >out`
- `awk -f cmdfile <in >out`
- program flow like in sed
- input automatically split in records and fields
- program is a list of blocks: `cond { commands }`

awk: features

- variables, assoc arrays
- functions
- pattern matching (EREs)
- dynamic typing
- more high-level: no pointers

awk: statements

much like C

- `if`, but no `switch-case`
- `while`, `do-while`
- `for (;;) {}` like in C
- `for (i in array) {}` foreach
- `break`, `continue`, `exit`, `return`
- `print` and `printf` are statements! redirection possible with >, >>, |

awk: functions

arithmetic	much like in C
I/O	`getline(), system()`

string	`sub(), gsub()`
	`substr(), index(), match(),`
	`split(), sprintf()`
	`tolower(), toupper(),`
	`length(), int()`

awk: variables

`RS, FS`	record/field separator
`NR`	number of current record
`NF`	number of fields (`$1` ... `$NF`)
`ARGC, ARGV`	like in C
`FILENAME`	name of current input file
`OFMT`	output format for numbers
`ORS, OFS`	output record/field separator
`SUBSEP`	`a[1,2]` equals `a[1 SUBSEP 2]`

awk: conditions

examples:

- `NR == 1 {...}`
- `NR == 1, NR == 5 {...}`
- `/RE/ {...}`
- `$1 == "string" {...}`
- `$1 ~ /RE/ {...}`
- `BEGIN {...}`
- `END {...}`

<awk demo>

real life examples

literature

- “**The Unix Programming Environment**” by Kernighan and Pike is highly recommended to support this talk
- it’s one of my favorite computer books
- you really should read it!

- “**SED & AWK ge-packt**” by Stephan Thesing
- if you want a german book about `sed` and `awk` then get this one
- you can get it for about 3 Euro

this talk was prepared using tools of the Heirloom project:
`http://heirloom.sf.net`
the slides macros were taken from
`http://repo.cat-v.org/troff-slider/`
all editing was done with `ed`, of course :-)

the slides and examples are available on my website
`http://marmaro.de/docs` and on
`http://ulm.ccc.de/ChaosSeminar/2009/10_Softwareschaetze`

2009-10-12