Introduction to Bash video lecture

11 - More on text files

File type versus file name

- In decent operating systems, the file name has no meaning
- You still use suffices to make it clear for humans the type of file
- The computer couldn’t care less
- Use `file` to investigate filetype

```
rikard@newdelli:~/bash-intro/text-files$ cat todo-list.txt
* Buy beer
* Throw party
* Clean apartment
rikard@newdelli:~/bash-intro/text-files$ mv todo-list.txt todo-list.mp3
rikard@newdelli:~/bash-intro/text-files$ cat todo-list.mp3
* Buy beer
* Throw party
* Clean apartment
rikard@newdelli:~/bash-intro/text-files$ file todo-list.mp3
todo-list.mp3: ASCII text
```

Structure of plain text files

- Text consists of characters, words, lines and paragraphs
- A word is printable characters separated by *whitespace* (blanks, tabs, newlines)
- A line is characters or words with a newline character at the end
- A paragraph is lines separated by an extra newline
- Everything is encoded in binary using e.g. the ASCII table for the numbers

US ASCII table

0	NUL	16	DLE	**32**		48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	**HT**	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	**LF**	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

Expressing whitespace in Bash etc

- A blank is written as is (press the space bar)
- A tab is *escaped* as \t
- A newline (linefeed) is escaped as \n

```
$ echo -e "Number\tName\tPrice\n9\tUrquel\t12.90"
Number  Name    Price
9       Urquel  12.90
$

# -e is the flag to echo for using escaped characters
```

How many characters?

```
These are four words.
```

How many characters?

```
These are four words.
```

- Depends! Is there a newline at the end?
- It is common to end a text with a newline (but not required)

ASCII-encoded

```
 84 (T)
101 (e)
115 (s)
101 (e)
 32 (blank)
 97 (a)
114 (r)
101 (e)
 32 (blank)
102 (f)
111 (o)
117 (u)
114 (r)
 32 (blank)
119 (w)
111 (o)
114 (r)
100 (d)
115 (s)
 46 (.)
 10 (LF - newline)      ← Optional
```

Using wc to count text

- `wc -l` number of lines (actually, number of newlines)
- `wc -c` number of bytes
- `wc -m` number of characters
- `wc -w` number of words

Swedish characters take two bytes (are encoded using utf-8)

English characters take one byte (encoded as plain text ASCII)

Textwrapping

- Plain text is just lines (where empty lines are just a single newline)
- Lines have a width (the number of characters before the next newline)
- Applications displaying text will automatically wrap lines, so that the window displays the maximum number of words per line
- Changing the width of the application window will make the text look different
- Think about where you put newlines
- Either, newlines only occur between sections/paragraphs
- or, you will use a fixed linewidth

Using fold to create a fixed linewidth

```
rikard@newdelli:~/bash-intro/text-files$ fold -s --width=80 lorem.txt > lorem80.txt

## -s means “break at spaces” (don’t cut in the middle of a word)

## using wc to find lenght of the widest line

rikard@newdelli:~/bash-intro/text-files$ wc -L lorem80.txt
80 lorem80.txt
rikard@newdelli:~/bash-intro/text-files$
```

Fixed width helps when investigating text

- Text commands are line-based
- If lines are very long, results may be overwhelming
- Using, e.g. `grep` to print lines with some pattern you search for works not so well if your file has only one very long line - the whole file will be printed it there's a match

<<last page>>