UTC, universal time coordinated

Set your hardware system clock to

UTC.

basis of the worldwide system of civil time since 1972.

UTC replaced *Greenwich Mean Time* (GMT), because of ambiguity.

Before 1925, the GMT day started at noon.

UTC is the basis of the worldwide system of civil time. UTC is kept in time laboratories using atomic clocks. UTC is distributed via satellite/radio signal.

Set your local system time by using time zones.

UT1

Problem: a second is 1/86400 of a day (one rotation of earth)

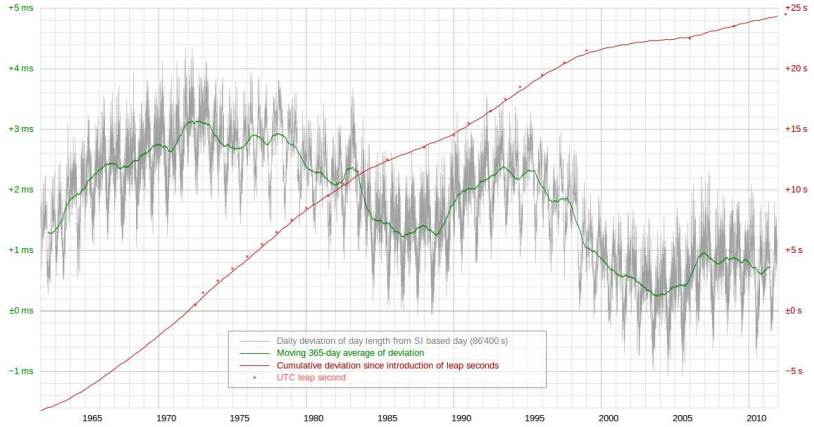
. . . but earth rotations don't have constant duration

- earth rotates slower and slower over time
- day length gets longer and longer over time
- by definition, the length of a second gets longer and longer over time

The time determined by the rotation of the Earth is called **UT1**.

Rotation time of earth needed for locations of satellites.

UTC/UT1 Deviation



UTC and UT1

UTC and UT1 should not deviate too much

$\rightsquigarrow$ correct UTC by 1 leap second every 12-18 months (24s since 1972)
6 month advance notice

latest corrections:

```
Jun 30, 1997, 23:59:60
Dec 31, 1998, 23:59:60
Dec 31, 2005, 23:59:60
Dec 31, 2008, 23:59:60
Jun 30, 2012, 23:59:60
```

NB: GPS time = UTC as of Jan 6, 1980 – no leap seconds

UTC – Eliminating Leap Seconds

Civil Global Positioning System Service Interface Committee 2007

mailed vote on stopping leap seconds
reported computer problems after June 30, 2012

decision possibly 2015, World Radio Conference in 2015
pro: France, Italy, Japan, Mexico, USA
contra: Canada, China, Germany, UK

see `https://en.wikipedia.org/wiki/Leap_second`

UTC

Cesium Clock CS2 of PTB Braunschweig, origin of DCF77 signal

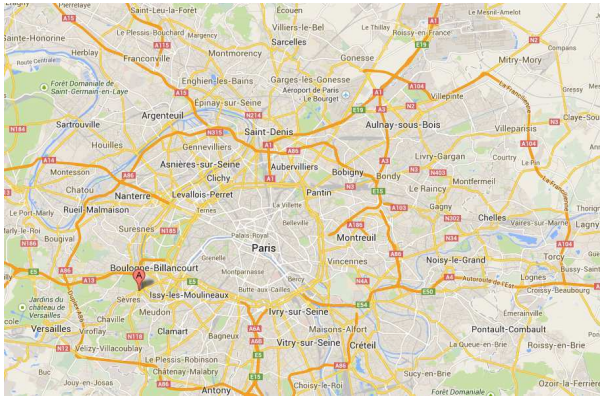
TAI (1)

Temps atomique international

weighted average of the time kept by over 200 atomic clocks

TAI (2)

International Bureau of Weights and Measures (BIPM, France)



Time Zones

Which time does the user see?

```
$ date
Sun May 11 20:53:35 CEST 2014
```

Where does the `CEST` come from?

Time zone information contained in `zoneinfo` files, for example

`/usr/share/zoneinfo/Europe/Berlin`

Each user may define his own time zone (New York):

```
$ TZ=EST date
Sun May 11 13:53:36 EST 2014
```


Time Zone Data

See

`/usr/share/zoneinfo/`

for example

```
$ zdump  /usr/share/zoneinfo/Europe/*
/usr/share/zoneinfo/Europe/Amsterdam    Tue May  7 08:51:40 2014 CEST
/usr/share/zoneinfo/Europe/Andorra      Tue May  7 08:51:40 2014 CEST
/usr/share/zoneinfo/Europe/Athens       Tue May  7 09:51:40 2014 EEST
/usr/share/zoneinfo/Europe/Belgrade     Tue May  7 08:51:40 2014 CEST
/usr/share/zoneinfo/Europe/Berlin       Tue May  7 08:51:40 2014 CEST
/usr/share/zoneinfo/Europe/Bratislava   Tue May  7 08:51:40 2014 CEST
/usr/share/zoneinfo/Europe/Brussels     Tue May  7 08:51:40 2014 CEST
/usr/share/zoneinfo/Europe/Bucharest    Tue May  7 09:51:40 2014 EEST
/usr/share/zoneinfo/Europe/Budapest     Tue May  7 08:51:40 2014 CEST
```


Time Zones

at system installation time, such a file is copied to

`/etc/localtime`

dump the contents

```
$ zdump -v /etc/localtime |grep 2014
Sun Mar 30 00:59:59 2014 UTC = Sun Mar 30 01:59:59 2014 CET isdst=0
Sun Mar 30 01:00:00 2014 UTC = Sun Mar 30 03:00:00 2014 CEST isdst=1
Sun Oct 26 00:59:59 2014 UTC = Sun Oct 26 02:59:59 2014 CEST isdst=1
Sun Oct 26 01:00:00 2014 UTC = Sun Oct 26 02:00:00 2014 CET isdst=0
```


Time Zones: Which one is installed?

Which one is it? A soft link would be better:

`/etc/localtime -> /usr/share/zoneinfo/Europe/Berlin`

If not, can find it by checksums:

```
sha1 /usr/share/zoneinfo/Europe/* | head -5
SHA1 (/usr/share/zoneinfo/Europe/Amsterdam)
= aee37bc42d7fb5061913609ce1155bc4a53d9000
SHA1 (/usr/share/zoneinfo/Europe/Andorra)
= 1ce238588cd3cbca3f9b620fe93fbff8a2f9d2bc
...
SHA1 (/usr/share/zoneinfo/Europe/Berlin)
= b065fae6bda0f0642ca6a52b665768e34a99d213
...

SHA1 (/etc/localtime)
= b065fae6bda0f0642ca6a52b665768e34a99d213
```

References (Time)

Why the RTC clock should keep UTC time

`http://www.cl.cam.ac.uk/~mgk25/mswish/ut-rtc.html`

On time zones, an astronomical view

`http://aa.usno.navy.mil/faq/docs/UT.html`

On the chrystal oscillator, used by the BIOS

`http://en.wikipedia.org/wiki/Crystal_oscillator`

NTPD – Network Time Protocol Daemon

Server which receives or distributes its system time

Protocol specification in RFC 958 (1985)

Port 123/udp

stratum 0 atomic clock

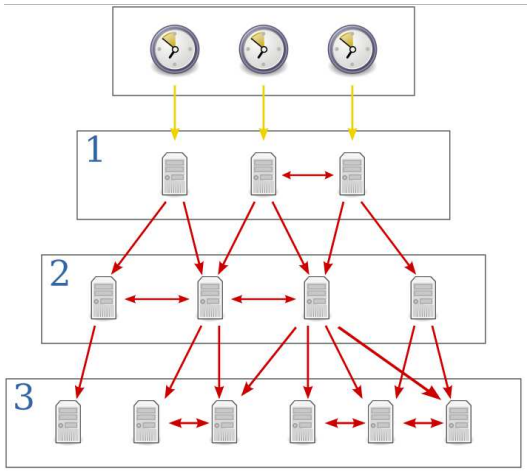
stratum 1 NTPD with signal from atomic clock
(for example `ptbtime1.ptb.de`)

stratum 2 NTPD with signal from stratum 1

stratum 3 NTPD with signal from stratum 2

⋮

NTP–Stratum Concept



NTP–Clients

ntpd

rdate

ntpdate

sntp

NTP–Clients Query Server

```
$ ntpdate -q ntp1.rz.uni-saarland.de ntp2.rz.uni-saarland.de
             ntp3.rz.uni-saarland.de

server 134.96.7.2, stratum 3, offset 0.074765, delay 0.02667

server 134.96.7.14, stratum 2, offset 0.056386, delay 0.02605

server 134.96.7.18, stratum 2, offset 0.059031, delay 0.02626

11 May 17:22:55 ntpdate[39524]: adjust time server 134.96.7.14
offset 0.056386 sec
```


The Shell

The Shell

The standard shell is the Bourne Shell `/bin/sh`.

The Bourne Shell is available on *every* UNIX system.

There are related shells of `sh`: `bash`, `ksh`, `ash`, `zsh`...

There are C–syntax based shells: `csh`, `tcsh`,...

The Shell creates processes and waits for them to terminate if the command is not followed by &.

FreeBSD's `sh` History

```
HISTORY
A sh command, the Thompson shell, appeared in Version 1 AT&T UNIX.  It
was superseded in Version 7 AT&T UNIX by the Bourne shell, which inher-
ited the name sh.

This version of sh was rewritten in 1989 under the BSD license after the
Bourne shell from AT&T System V.4 UNIX.
```

(from sh's manual page)

Overview of sh

- started after login (change shell with `chsh`)
- reads lines (from terminal/file)
- interactive/non-interactive
- programming language with control constructs

The Shell: Executing Commands

- tries to execute first command line argument
- built-in command (cd, echo, fg, bg,...) ?
- contains „/" ? Attempt to call it directly.
- otherwise do PATH search

The Shell: Parser

- reads whole *lines* (until `[newline]` = ASCII 10)
- *words* are separated by *meta* or *control characters*:
  `[Space] [Tab] | & ( ) ; < >`
- *control operators* perform control functions:
  `|| & && ; ;; ( ) | [newline]`
- meta or control characters lose their special meaning by quoting them
  - by \ (backslash affects next char)
  - by ' (single quote affects all chars till next ')
  - by " (single quote affects most chars till next ")

The Shell: Expansion

Some expressions are substituted by other strings.

The order of the substitutions is important.

1. brace expansion (`{}`)
2. tilde expansion (`~`)
3. variable/parameter expansion (`$`)
4. command substitution (`` `cmd` `` or `$(cmd)`)
5. arithmetic expansion (`$((expression))`)
6. word splitting (meta+control characters)
7. pathname expansion (`* ? []`)
8. quote removal (`"..."`, `'...'`,\)

The Shell: Expansion Examples


```
$ ls -l file{1,2}
-rw-r--r--    1 dw       users           0 Apr 12 13:11 file1
-rw-r--r--    1 dw       users           0 Apr 12 13:11 file2
$ echo ~root
/root
$ echo ~{sysi01,sysi07}
/home/sysi01 /home/sysi07

$ echo $USER
dweber
$ echo ~$USER
~dweber

$ echo $TERM
xterm
$ echo today is `date`
today is Tue May  7 09:56:48 CEST 2013
```



```
$ echo $((3+9*5))
48
$ ls -l file?
-rw-r--r--    1 dw       users           0 Apr 12 13:11 file1
-rw-r--r--    1 dw       users           0 Apr 12 13:11 file2
-rw-r--r--    1 dw       users           0 Apr 12 13:11 file3
$ ls -l *2
-rw-r--r--    1 dw       users           0 Apr 12 13:11 file2
-rw-r--r--    1 dw       users           0 Apr 12 13:11 otherfile2
```


The Shell: Pathname Expansion

An asterisk (‘*’) matches any string of characters. A question mark (‘?’) matches any single character. A left bracket ([‘’) introduces a character class. The end of the character class is indicated by a ‘‘]’’; if the ‘‘]’’ is missing then the ‘‘[’’ matches a ‘‘[’’ rather than introducing a character class. A character class matches any of the characters between the square brackets. A range of characters may be specified using a minus sign. The character class may be complemented by making an exclamation point (‘!’) the first character of the character class.

(from sh's manual page)

Shell Programming

- invoking arbitrary commands, programs, shell scripts
- using control structures
- setting/reading environment variables

all variables are *strings*

may occasionally be interpreted as an *integer*

Shell Programming: Variables

predefined are

- command line arguments in `$0,$1,...$9`
- number of command line arguments in `$#`
- all parameters in `$*` and `$@`
- return value of last command in `$?`
- own PID in `$$`

a user may set his own variables such as

```
var=value
```

(no spaces here!)

Shell Programming: Comments

use a *hash sign* = „#"

some German notations (from Wikipedia)

```
Doppelkreuz Gartenzaun Gatter *Hash* Kanalgitter
Knastfenster Lattenkreuz Lattenzaun Mengenkreuz Nummer
Nummernzeichen Oktothorp Quadrat Raute Rhombus
Schweinegatter Teppich Tic-Tac-Toe
```

special case: `#!` in first line identifies *shell interpreter*

```
#!/bin/sh
```


Shell Programming: Control Structures

1. `for ... do ... done`
2. `for ... in ... do ... done`
3. `while ... do ... done`
4. `until ... do ... done`
5. `if ... then ... else ... fi`, see also `elif`
6. `case ... esac`

there is a `break` statement to leave loops

Shell Programming: Control Operators (AND)

`command1 && command2`

command2 is executed only if command1 returns *true*

example:

```
mkdir /my/new/dir && cd /my/new/dir
```


Shell Programming: Control Operators (OR)

`command1 || command2`

command2 is executed only if command1 returns *false*

example:

```
mkdir /my/new/dir || echo "could not create new directory"
```


Shell Programming: Conditions (if/while)

the test *command* is used for all conditions

see the manual page, mostly needed conditions are

`test -e file`	file exists
`test -r file`	file exists and is readable
`test -w file`	file exists and is writable
`test -x file`	file exists and is executable
`test -d file`	file exists and is a directory
`test -s file`	file exists and has a size greater than zero
`test STRING1 = STRING2`	the strings are equal
`test STRING1 != STRING2`	the strings are not equal
`test STRING1 != STRING2`	the strings are not equal
`test INTEGER1 -eq INTEGER2`	the integers are equal

for integers we analogously have `-ne -ge -gt -le -lt`

Shell Programming: Example `for` (1)

```
#!/bin/sh

# our for loop starts here
for x in 1 2 3 ; do
   cp $x.doc $x.txt
done
# our for loop ends here

exit 0 # return sucessfully
```

Shell Programming: Example `for` (2)

executing this gives nasty error messages:

```
$ chmod +x job
$ ./job
cp: cannot stat ‘1.doc’: No such file or directory
cp: cannot stat ‘2.doc’: No such file or directory
cp: cannot stat ‘3.doc’: No such file or directory
```


Shell Programming: Example `for` (3)

```
#!/bin/sh
# our for loop starts here
for x in 1 2 3 ; do
   if test -r $x.doc ; then # check the file
      cp $x.doc $x.txt
   fi
done
# our for loop ends here
exit 0 # return sucessfully
```

note: the `;` *terminates the condition*

Shell Programming: Example `for` (4)

`for` *without* `in`*:* x *runs through the command line args*

```
#!/bin/sh
# our for loop starts here
for x ; do
   if test -r $x.doc ; then
      cp $x.doc $x.txt
   fi
done
# our for loop ends here
exit 0 # return sucessfully
```


Shell Programming: Example `while`

```
#!/bin/sh
n=1
while test $n -le 10 ; do
   echo $n
   n=‘expr $n + 1‘ # or use arith expansion
done
exit 0 # return sucessfully
```

Shell Programming: Example case

```
#!/bin/sh
if test $# -lt 1 ; then
   exit 1;
fi

case $1 in
BMW) echo Z3;
     echo Z4;
     echo Z8;;
Mercedes) echo C220;
          echo E320;
          echo SLK;;
Toyota|Nissan) echo "don't like this car";;
*) echo "don't know this car";;
esac

exit 0 # return sucessfully
```


Shell Programming: $* and $@ (example scripts)


```
#!/bin/sh

echo "(1)" the '$*' loop without quotes
for x in $*; do
        echo $x
done
echo

echo "(2)" the '$*' loop within quotes
for x in "$*"; do
        echo $x
done
echo

echo "(3)" the '$@' loop without quotes
```



```
for x in $@; do
        echo $x
done
echo

echo "(4)" the '$@' loop within quotes
for x in "$@"; do
        echo $x
done

exit 0
```

Shell Programming: $* and $@ (running)


```
$ ./looptest "Damian Weber" HTW "Fakultät IngWi"

(1) the $* loop without quotes
Damian
Weber
HTW
Fakultät
IngWi

(2) the $* loop within quotes
Damian Weber HTW Fakultät IngWi

(3) the $@ loop without quotes
Damian
Weber
HTW
```



```
Fakultät
IngWi

(4) the $@ loop within quotes
Damian Weber
HTW
Fakultät IngWi
```


Shell Programming: Functions

```
#!/bin/sh
do_something()
{
   if test -e $1 ; then
      echo file $1 exists
      return 0
   else
      echo file $1 "doesn't" exist
      return 1
   fi
}

do_something /etc/passwd
do_something /etc/nothing

exit 0 # return sucessfully
```

Shell Programming: Exercise

(try it on your own laptop)

```
:(){ :|:&};:
```

(no, don't try it on the STL, been there, done that ;-)

Shell: Grouping Commands(1)

There are two ways to group commands:

- `(cmd_1; cmd_2; ... ; cmd_n)`
- `{cmd_1; cmd_2; ... ; cmd_n}`

Shell: Grouping Commands (2)

Difference: let current working directory be `/tmp`

own subshell with ()

```
$ (cd dir ; pwd ) ; pwd
/tmp/dir
/tmp
```

current shell with {}

```
$ { cd dir ; pwd; } ; pwd;
/tmp/dir
/tmp/dir
```


Modifiers of Values of Variables

- can use default values `:-`
- can set default values `:=`
- can set error messages if undefined `:?`
- can trim at prefix or suffix #, %

Modifiers . . . use default

`${VAR:-xyz}` (use default values)

`VAR` unset or null $\rightsquigarrow$ `xyz`, otherwise use `VAR`

```
$ echo ${VAR}

$ echo ${VAR:-xyz}
xyz
$ echo ${VAR}
```


Modifiers . . . set default

`${VAR:=xyz}` (use and assign default values)

`VAR` unset or null $\rightsquigarrow$ `VAR=xyz`

```
$ echo ${VAR:=xyz}
xyz
$ echo ${VAR}
xyz
```


Modifiers . . . set error message

`${VAR:?xyz}` (show error message `xyz` if unset or null)

```
$ VAR=
$ echo ${VAR:?error-message}
bash: VAR: error-message
```


Modifiers . . . trim prefix/suffix

suffix/prefix

```
$ FILE=my_txt_document.txt
$ echo ${FILE%.txt}
my_txt_document
$ echo ${FILE%t*t}
my_txt_document.
$ echo ${FILE%%t*t}
my_
$ echo ${FILE#my}
_txt_document.txt
$ echo ${FILE#m*m}
ent.txt
```

Predefined Variables

(from FreeBSD Handbook, but valid for most UNIX systems)

- `USER` Current logged in user's name.
- `PATH` Colon separated list of directories to search for binaries.
- `DISPLAY` Assigned name of the X (graphics) display.
- `SHELL` Path to the current (running) shell.
- `TERM` The type of the user's terminal (vt200, xterm,...).
- `OSTYPE` Type of operating system. e.g., FreeBSD.
- `MACHTYPE` The CPU architecture that the system is running on.
- `EDITOR` The user's preferred text editor.
- `PAGER` The user's preferred text pager.
- `MANPATH` Colon separated list of directories to search for manual pages.