I. History Milestones for Computers, Unix and the Internet

Unix is a time-sharing (multi-user, multi-tasking) operating system, whose development began in the 1960s. Unlike many operating systems from this era, Unix is still thriving and growing. It is the operating system of choice in many super-computers and high-end engineering workstations. It is finding increasing use in personal computers, thanks in large part to the popular Linux version of Unix developed to run on Intel processors. Its most important use in many business situations is to support database, Web and most recently cloud servers.

Unix remains popular for several reasons. It is portable. It has a flexible architecture. It is written in a high-level language (C), and hence can be easily implemented on almost any hardware platform. The very age of Unix has made it reliable, as most of the bugs were fixed long ago. Unlike Windows, which was originally designed to run several tasks intermittently for a single user, Unix was designed to run several tasks simultaneously for hundreds of users. In doing so, it's designers implemented techniques for sharing disk space, processor time, and other machine resources among hundreds of time-sharing users. As needs and applications changed, these same resource-sharing techniques provided an efficient mechanism to service the needs of thousands of simultaneous users of database and Web servers. Finally, the Unix hierarchical file system, developed to share disk space across multiple disk drives among multiple users, is reflected in the Internet URL naming conventions. A similar file system philosophy is used in nearly all modern operating systems.

A brief, "milestone" history of Computers, Unix and the Internet follows.

Circa 3000BCE, the abacas is invented somewhere in Asia - perhaps China or India.

1642 • Blaise Pascal designs a mechanical calculator (not successfully built).

1673 • Wilhelm Leibniz designs an improved calculator (not successfully built).

1801 • Joseph-Marie Jacquart uses punch cards to control weaving patterns in a loom.

1829 • William Austin Burt patents the first typewriter.

1850 • Charles Babbage designs a mechanical computer for logical and arithmetic calculations. It was somewhat successfully built with steam-power.

1850-1940 Symbolic logic is created by Boole, Frege, Peano, Russell, Godel, Turing, et al.

1890 • Herman Hollerith invents punch card computing devices to analyze the US Census.

1920 • Karel Capek's play, "RUR", includes the first use of the word "robot".

1931 • Reynold B. Johnson invents the scantron device.

1939 • John Atanasoff builds a successful electro-mechanical computer (using vacuum tubes).

1943 • Alan Turing and other UK scientists, working to break the German Enigma code, build Colossus, the first all electronic computer.

1946 • Presper Eckert, John Mauchly and other U. of Penn. scientists build ENIAC, the first general purpose electronic computer.

(The above 3 computers were externally programmed using "plugboard patch-panels".)

1947 • Bardeen, Brattain and Shockley of Bell Labs invent the transistor.

1949 • EDVAC, the first general purpose, stored program computer is built at U. of Penn, based upon the stored program design of John von Neumann.

1949 • John Mauchly develops Short Order Code, arguably the first programming language.

1950 • Alan Turing conceives the idea of the "Turing Test" for Artificial Intelligence.

1950's: The Computer Age begins. Successful computers using transistors and semi-conductors are marketed by IBM, Honeywell, RCA, Sperry, Univac, Burroughs, etc. The use of electronic computers becomes pandemic in science, engineering, and big business. Computers are connected in various experimental ways to facilitate transmission of data.

1951 • Maurice Wilkes conceives of "micro-programming", which makes designing operating systems easier.

1953 • The IBM 650 is marketed as arguably the first successful programmable electronic computer for business. The IBM 701, 1400 and 1620 soon follow.

1954 • John Backus invents FORTRAN, the first high-level programming language.

1956 • Marvin Minsky and John McCarthy, at a Dartmouth college meeting, coin the phrase AI for Artificial Intelligence.

1957 • John McCarthy creates MIT's AI Department, the world's first.
• Grace Hopper of the USN invents COBOL, the first "user-friendly" programming language for business applications.

1958 • Project Whirlwind creates the first "real time" computer system (for air traffic).
• Bell Labs develops the modem data phone.
• In the wake of Sputnik, The Defense Advanced Research Projects Agency (DARPA) is established to fund strategic research.

1959 • GE invents the first check processing machine.

1960 • DEC releases the PDP-1, the first commercial computer with both typewriter and monitor-keyboard consoles.

1961 • Leonard Kleinrock, at MIT, conceives the notion of packet switched networks of computers.

1962 • Stanford and Purdue establish the first two departments of Computer Science.
• Joe Licklider, author of "Man-Computer Symbiosis"(1960), is hired by DARPA.
• The first computer video game is invented at MIT by Steve Russel.

1963 • IBM announces the System-360 family of computers.
• Ivan Sutherland introduces Sketchpad (the first drawing program).

1964 • Doug Engelbart invents the mouse.

1965 • Donald Davies creates the first practical packet-switched network protocol.
• Ted Nelson invents hypertext.

1966 • DAPRA authorizes Bob Taylor and Charles Herzfeld to build DARPANET, linking the universities funded by DARPA.
• ATT's Bell Labs and MIT begin a joint effort to develop MULTICS, a time-sharing operating system for scientific applications.

1967 • Wesley Clark and Larry Roberts invent the concept of an IMP - Interface Message Processor (arguably the first router).

1968 • A small Massachusetts consulting firm, Bolt, Beranak and Newman (BBN) wins the DARPA contract to build DARPANET for $1 million. Honeywell builds much of the hardware. IBM and ATT didn't even bid, as they thought the concept impractical.
• MULTICS partnership is discontinued, but Dennis Ritchie and Ken Thompson, working for Bell Labs, continue to work on the concept.

1969 • Thompson implements the still unnamed Unix on a Digital PDP-7 computer. Written in machine language, it runs his orrery and attracts a following among other Bell scientists.
• The first ARPANET site is installed at UCLA by BBN. It is linked to subsequent sites at Stanford, Santa Barbara and Utah.

1970 • Xerox establishes the Palo Alto Research Center (PARC), which will be the source of many key computer innovations.
• Steve Crocker designs NCP (Network Control Protocol) to improve DARPANET performance.
• Ritchie implements a new programming language, called "C", on a PDP-11 computer.

1971 • Brian Kernighan is named manager of the Bell labs department where Thompson and

Ritchie work. He encourages further work on "C" and Thompson's operating system, which Kernighan names UNIX.

1972
- Ray Tomlinson at BBN designs the first email.
- Jon Postel designs Telnet.
- Vinton Cerf is named chairman of Inter-Networking Group.
- Wang introduces the first successful commercial word processing system.

1973
- Robert Metcalfe "invents" Ethernet.
- Ken Thompson and Dennis Ritchie rewrite their operating system in C and UNIX is born. (See: http://cm.bell-labs.com/cm/cs/who/dmr/chist.html)
- PARC developes the ALTO personal computer, with GUI monitor, mouse and Ethernet LAN. It supports Bravo, the first WYSIWYG application, written by Charles Simonyi.
- DARPANET supports 100 sites, including the College of London.

1974
- Vinton Cerf and Bob Kahn design TCP, a precursor to TCP/IP, the network protocol of the Internet. In their paper, they coin the word "Internet".
- UNIX is licensed to educational institutions, creating the Open Source code philosophy.

1975
- Jon Postel warns that email may lead to massive junk mail abuse (spamming).

1976
- The UUCP standard, built on top of TCP and Unix, is developed at ATT.
- Steves Jobs and Wozniak introduce The Apple I.
- Al Gore is elected to Congress.

1977
- PARC enhances ALTO into a practical GUI (graphical user interface).
- The Tandy and Commodore PCs, with built in monitors, are introduced.

1978
- Berkeley, a key Internet site, publishes the first open source code version of UNIX.
- Cerf, Crocker and Dan Cohen create specifications for the full TCP/IP protocol.
- TCP/IP is first implemented using computers running Unix. Until the 1990's, most Internet hosts would be Unix computers.

1979
- The USENET is created by Tom Truscott, Steve Bellovin and Jim Ellis.

1980
- ATT releases its 2B computer, using Unix. ATT begins using its own computers, running Unix, to support all of its new PBX (Private Branch Exchange) products.

1981
- The IBM PC and MS/DOS are introduced.
- PARC introduces GUI applications with clickable icons and Windows. (But since Xerox is "The Document Company" they see no future business potential. Steve Jobs does and incorporates many of the ideas into the MacIntosh.)

1982
- All 200+ Internet sites are converted to the TCP/IP protocol.
- Al Gore is elected to the Senate.

1983
- Berkeley UNIX 4.2/BSD standardizes important UNIX and TCP/IP network conventions.
- Significant enhancements in Internet capacity are proposed under the name NFSNET.
- Apple introduces the MacIntosh and the first GUI commercial applications.

1984
- The number of Internet hosts exceeds 1000.
- William Gibson writes the novel Neuromancer and invents Cyberspace.

1985
- Microsoft releases Windows 1.0 (a largely unsuccessful imitation of MacOS).

1987
- Apple Computer's Hypercard becomes the first end-user authoring system for hypertext.

1989
- Tim Berners-Lee, working at CERN in Europe, invents the "World Wide Web".

1990
- Microsoft releases Windows 3.0 (which almost works), soon followed by Windows 3.1.
- Alan Emtage invents Archie, arguably the first search engine (he didn't patent it).

1991
- Mark McCahill creates the Gopher search engine.

1993
- Marc Andreessen and Eric Bina create MOSAIC, the prototype for today's browsers.
- Excite launches the first commercial search engine.

1995
- Jeff Bezos launches Amazon.com.
- Pierre Omidyar launches ebay.com.

1997
- The FBI implements Carnivore to spy on all email and electronic communications.

1998
- Larry Page and Sergey Brin launch Google, with its revolutionary business model.

2001
- Jimmy Wales and Larry Sanger launch Wikipedia.

II. Overview of the Unix Operating System

The **Unix operating system** presents a four-layer view of the computer: hardware, kernel, shell, applications. At the center is the **hardware**. Unix surrounds the hardware with a program called the **kernel**, which acts as an standardized interface to the hardware and manages the sharing of these resources. Users communicate with the kernel through the **shell**. The shell is a program that accepts commands to run other programs, which are either **shell commands** or **applications**. When a program is run, the shell requests the kernel to **fork** or **spawn** a new **process**, which is a region of memory into which the program is loaded and executed. Each process has a Process ID (PID) which uniquely identifies it. Information about processes is displayed using the **"ps"** command. Each process has a parent process - the process which spawned it - whose PID is called the PPID of the child process. The processes thus form a hierarchy.

The Command Line

The shell command line uses a standard syntax for all commands:

Command [-options] [arguments] [< input , > output , | , &]

Information on the particular options and arguments for each command may be found with the **"man"** command - an online help manual. A command may accept input from the keyboard called **standard-in**. A **"<"** redirects standard-in to come from a file. A command may display output on the screen called **standard-out**. A **">"** redirects standard-out to go to a file. A **"|"** or **pipe** connects standard-out of one command to standard-in of another. A **"&"** causes execution of the command in the background.

Some basic commands are:

login	login to Unix host (**rlogin** for remote login)
exit	exit from Unix
passwd	change password
date	display the date and time
cal	display a perpetual calendar
who	display users currently logged in **(see who am i)**
ping	display user connection (see **finger**)
hostname	display name of host Unix server
mail	Unix email program
man	online help manual
clear	clear screen
^c	kill the current process
^d	end keyboard input
ps	display process status (see: **ps –f**)
exec	execute another program without forking
kill	kill a process
fg,bg	manage background processes

The File System

Unix stores all files in an unified, **hierarchical (branching, multi-level) file system**. This implementation insulates the user from any knowledge of the specific, physical storage devices. Each level of the system is a **directory**. Each such directory contains two types of entries - **files** and **sub-directories**. Sub-directories are pointers or "links" to other directories, which themselves contain files and sub-directories. Files and directories are uniquely named by the chain of sub-directories under which they exist, plus a file name for the file in question. The fully qualified name is called the **absolute pathname**. When one is logged into Unix, one always has a **present working directory**, where all shell commands usually look for files by default. The command **pwd** displays the present working directory. Files named relative to the present working directory are called **relative pathnames**.

The top level of the hierarchy is called **root** and symbolized as **"/"**. It contains a number of files, among them the executable program, which is the Unix kernel. Root also contains a number of standard sub-directories:

/usr - contains sub-directories of commands and application programs.
/usr/bin - contains most of the shell commands (/usr/bin/sh is the Shell).
/usr/sbin - contains other shell commands used mostly for system administration.
/etc - system administration and configuration files (/etc/passwd has passwords).
/dev - contains files which "represent" all peripheral hardware devices.
/home - usually contains one directory for each user on the system.
/tmp - an area for temporary files often needed by programs (e.g. for piping).

The basic shell commands to move around the file system and work with files are:

cd changes present working directory (see also **pwd**)
ls lists the contents of a directory (see options "ls -l" and "ls -al")
cat displays a file
cp copies a file (**rcp** copies files between different hosts)
find finds files, returning full path name
ftp transfers files between different machines
lp print a file (see also **lpstat** and **cancel**)
mv moves or renames a file
od displays a file in octal or hex
rm removes a file
sort sorts a file
diff compares and displays differences between two files
spell spell check a file
mkdir creates a subdirectory
rmdir removes a subdirectory
chmod modifies permissions for a file (see also **chgrp**)

When supplying a file name to these commands, one may use wildcard characters to specify more than one file: "*" means any string of characters, "?" means any single character and "[...]" any in a list of characters.

The "ls -l" command displays information about a file:

type, permissions, links (see below), owner, group, size, date last modified, name

The **permissions of a file** determine whether a given user can read, write or execute the file. These permissions are based upon a "security philosophy" where every user is assigned to a group, and security is granted in three levels: single user owner of the file, user group of the file, everyone.

If a file name begins with a **"."**, it is a hidden file, like your **".profile"** script. Information about such files may be displayed using the **ls –al** command. In particular, every sub-directory contains two files:

"." - a file containing the name and location each file in the sub-directory

".." - a file containing a link to the parent directory

Each file is uniquely identified by a pointer to its location on a physical hardware device. The pointer is called an **inode**. Because a directory does not "contain" its files, but merely a list of file names and inodes, it is possible for a single file to have multiple names in multiple directories. When this is done, each "name" is just a separate entry in a directory **"."** file. To create additional names or links for an existing file, use the **ln** command.

ln old-name new-link-name

The above creates a physical link. It may be in the same or a different sub-directory, but it must be in the same file-system (see below). All physical links are equivalent - there is no primary physical link. To remove a physical link, use the **rm** command. When the last physical link is removed, the file (inode) is removed.

One may also create a symbolic link, which is permitted to cross different file-systems. The command is:

ln -s physical-link new-symbolic-link.

A symbolic link is dependent upon, and secondary to, the physical link. If the physical link is removed, the symbolic link becomes inaccessible.

The **df** command displays the physical file-systems which support the Unix file storage methodology. Each file-system may be thought of as a separate disk drive, but rather than use drive letters like Windows, Unix specifies mount points within the file hierarchy. The primary file-system is called the root file system and is mounted at **"/"** - the root directory. Other file-systems are mounted (using the **mount** command) at specified sub-directories under root.

Booting a Unix System

When one "boots" or turns on a Unix computer, the following events occur:

Kernel is loaded into memory,
Root file-system is mounted,
The program **init** is executed to initialize the system and control logins.

The task runnning **init** is assigned PID=1.

The **init** program may be run in multiple modes or levels:

-s for single user mode ,
-2 for multi-user mode,
-0 to halt the computer.

The default level is specified in the file, **/etc/inittab** . This file also contains instructions to mount the other file-systems and to start various background processes, or daemons, necessary to provide various system services.

An important standard daemon is **ttymon** which monitors direct terminal logins from terminals. When activity is detected at a terminal port, **ttymon** notifies **init** . This causes **init** to spawn a new process and run, in order, without forking (see **exec** command):

getty to setup a terminal environment,
login to verify ID & password (/etc/passwd) and setup a shell environment,
shell one of /usr/bin/sh,ksh,csh.

Other standard daemons include **inetd** and **ftpd,** which monitor network login and file transfer requests. When a network login is requested, **inetd** spawns a telnetd daemon to handles the details. When a network file transfer is requested, **ftpd** performs a similar service.

The Super-User

User security and system setup and customization often differ for various Unix vendors. Sometimes a program named **sysadmin** provides a menu-driven utility to perform these function. Other times, one uses the standard functions - **useradd, userdel, usermod** - to change user security information. Access to these utilities is usually restricted to the root ID, which is also called the super-user or "su" ID. The password to this must be carefully guarded.

III. Unix Shell Commands and Scripts

All Unix files are stored under an integrated file-system. All Unix commands run in a common environment. As such, all I-O is essentially equivalent. Unix commands have three standard "file assignments": standard input = keyboard, standard output = screen, and standard error = screen. These standard input-output "file assignments" may be redirected:

"command < filename" : the input to a command will be from a file.

"command > filename" : the output from a command will be stored in a file.

"command >> filename" : the output from a command will be appended to a file.

"command1 | command2" : the output of command1 will be the input to command 2.

This last type of redirection is called **piping**. The symbol "|" is called a **pipe**. Connecting simple commands with a **pipe** to perform compound functions is the essence of the Unix philosophy. Some commands were designed to perform a simple function upon standard in and write it to standard out. They are called **filters**. Some common ones are:

grep	search for words
lp	print
more	display standard-in to standard-out pausing every page (also **pg, page**)
sort	sort by column or word position
tee	save standard-in to a file and display to standard-out
wc	count characters, words and lines

Various special characters used to control command execution and input-output redirection:

|	connect or pipe standard output of one program to standard input of another
<	redirect standard input from a file
>	redirect standard output to a file
>>	redirect standard output to append to a file
1>	redirect standard output to a file
1>>	redirect standard output to append to a file
2>	redirect standard error to a file
2>>	redirect standard error to append to a file
1>&2	redirect standard output to standard error (as well as 2>&1)
&	run this command in the background
;	run commands sequentially, just as if specified on separate command lines
:	Null command, returns "true".
\	ignore conventional meaning and treat as a character
"....."	define a single text string with variable names replaced by values
'.....'	define a single text string with variable names unreplaced
`.....`	interpret as a command with standard output returned as a text string

Shell Variables

Associated with the shell is an area of memory which contains the names and values of stored variables. Variables are created, or their values changed, by one of:

VARNAME=text **VARNAME="text"**

VARNAME='text' **VARNAME=`text`**

Varaibles are accessed by preceding the name with a "$". By convention, variable names are capitalized, but this is not mandatory and variable names are case sensitive. The **echo** command may be used to display the value of a variable and the **read** command to create one:

echo $VARNAME **read VARNAME**

A list of all current user-defined variables may be obtained using the **set** command. If one spawns a sub-shell, the variables in the parent shell are available for access by the sub-shell only if they have been "exported" with the **export** command. A list of all "exported" variables may be obtained using the **env** command.

Among the standard, exported variables are $HOME - your home directory - and $PATH - your program search path. There are also standard system defined variables:

$0	current script being executed
$1-$9	arguments to current script
$#	number of $1 thru $9 which are set
$*	all of $1 thru $9
$$	PID of current task
$!	PID of last background task
$?	Last returned exit code

NOTE: When using quotes in scripts or with commands, very different behaviors result from different choices of quotes. If double quotes are used ("....."), any shell variables are replaced by their values. If single quotes are used ('...'), shell variables are retained in symbolic form. Finally, if back quotes are used (`...`), the character string is executed as a Unix command and replaced by the standard output of the command. For example:

The command: **echo $HOME** or **echo "$HOME"** :: returns something like: **/home/userid**.

The command: **echo '$HOME'** :: returns: **$HOME**.

The commands: **echo date**, **echo "date"**, **echo 'date'** :: all return: **date**.

The command: **echo `date`** :: returns: **Tue 26 Sep 14:24:03 EDT 2000**.

Shell Scripts

A series of shell commands stored in a file can be executed as a program called a shell script. To do this, the following command must be executed to change the permissions on the file to mark it as executable:

chmod +x scriptname.

The following is a brief summary of Unix commands, which are often used in shell script programs. This list is followed with some examples of coding techniques.

cat	concatenates multiple inputs (or standard input) to standard output
clear	clears the terminal screen
cut	cuts only specified columns or fields
echo	displays text strings
env	displays exported variables
eval	evaluate an expression, re-substituting variable values
expr	evaluates a text expression, includes logical and arithmetic operators
fold	splits lines at specified column
grep	searches for a text string (special characters "^","$",".")
head	displays first n lines
paste	pastes multiple inputs into columns (side by side cat)
read	accepts terminal input into a variable
sed	performs specified edits - "s/..../...../"
set	displays variables or sets special variables $1 thru $9
sleep	pauses for some number of seconds
sort	sorts
tail	displays either last n lines (-n) or all lines after the first n (+n)
test	evaluate a logical expression
tr	translates from one text string to another
unset	removes a variable
wc	counts lines, words and bytes
xargs	passes each line of input as arguments to a command

Remember, if in doubt on how to use a given command, the "man" command provides an online help manual for each. To print the "man" information for a given command:

man command | lp

Common Coding Techniques

The Bourne shell provides a rich programming environment, including logical and numeric operators to test variables and modify program flow. (Other shells work similarly.)

To test the value of a variable and alter the logic flow of a shell script, use "if":

```
if test "$X" = "$Y"
then
        ....(commands)
else
        ....(commands)
fi
```

In the above, either $X or $Y can be a text string or a variable. There are various options, including "-f" to test for file existence, which may be used instead. Another format of the command is:

```
if [ "$X" -op "$Y" ]
then
        ....(commands)
elif [ "$X" -op "$Y" ]
then
        ....(commands)
else
        ....(commands)
fi
```

In the above, the logical operator "-op" may be "-eq", "-ne", "-gt", "-ge", "-lt", "-le", or "=". Also, multiple tests may be made, connected by "-o" for "or" or "-a" for "and".

Note: The **test** command is just a program which is executed, evaluates the "truth" of its "argument expression", and ends by setting an exit status: True is zero and False is non-zero. The "if then else fi" construct may be used to evaluate the exit status of any command.

Rather than string together a long list of "elif" statements, it is easier to use the "case" construction:

```
case "$X" in
value1)
        ....(commands) ;;
value2)
        ....(commands) ;;
*)
        ....(commands) ;;
esac
```

The "case" construction is very useful in creating menus.

Loops can be implemented in three ways: "while", "until" and "for".

```
while [test]                    ( or until [test] )
do
        ....(commands)
done


for X in argument-list
do
        ....(commands)
done
```

The **[test]** is a valid "test" statement as described for the "if" statement. One may code **"while :"** to do an infinite loop. A "while" loops while the [test] remains true; an "until" loops until it is true. A "for" loop runs the variable $X thru a predetermined list of values, either an explicit argument-list, or $1 thru $9 if no list is given.

If one needs to increment a counter variable for each pass through a loop, the following examples illustrate some useful techniques.

```
CTR=1
while :
do
        CTR=`expr  $CTR  +  1`
done


CTR=1
for X
do
        echo arg-$CTR is $X
        CTR=`expr $CTR + 1`
done


CTR=1
while [ "$CTR" -le "$#" ]
do
        echo arg-$CTR is `eval echo $"$CTR"`
        CTR=`expr $CTR + 1`
done
```

Vi Text Editor

To create and edit shell scripts and other text files, you will use the **vi** text editor. A text editor is like a primitive word processor; and, as in a word processor, you can move a "cursor" around the file and insert, modify and delete text characters.

But unlike most word processors, **vi** uses no function keys. It works by toggling back and forth between command mode and input mode. When in input mode, whatever you type is inserted into the document. Hitting escape will place you in command mode. When in command mode, the following commands are useful:

Modes:

i - places you in input mode before the cursor position
a - places you in input mode after the cursor position
A - places you in input mode at the end of the current line
esc – return to command mode

:w - saves
:w filename - saves to a file
:q! - quits without saving

Cursor movement:

0 - go to beginning of cursor line
$ - go to end of cursor line
h,j,k,l - left, down, up, right arrows for cursor movement
w/b - moves forward/backward one word

:n - go to line n
:$ - go to end of file
:/...../ - searches for a text string

Editing:

x - deletes character at cursor
dw - deletes word at cursor
r - overtypes character at cursor
cw - overtypes current word
dd - deletes cursor line (and puts it into the paste buffer)
yy - copies cursor line into the paste buffer
(Each of the above may be preceded by a number to expand its scope of action.)
p - inserts the paste buffer after the cursor.
u - undoes last change

:n,m/...../...../ - replaces a text string with another (usually 1,$/...../...../)

The English Gregorian Calendar

The following is a shell script to "loop" the "cal" command thru every year and month since 1500 in order to determine when the conversion from Julian to Gregorian calendars occurred. This event included dropping over week from the calendar. The following will work:

```
clear
## looping thru cal to find short month
YR=1500
while [ $YR -le 2000 ]
do
        echo $YR
        for MTH in 1 2 3 4 5 6 7 8 9 10 11 12
        do
                set `cal $MTH $YR | tail +3 | wc`
                if [ $2 -le 27 ]
                then
                        cal $MTH $YR
                        exit
                fi
        done
        YR=`expr $YR + 1`
done
```

The result:

```
        Sept 1752

        01.02.14.15.16
 17.18.19.20.21.22.23
 24.25.26.27.28.29.30
```