One Too Many Shells ?

"I know engineers - they love to change things" - Dr. Leonard McCoy

1970's: sh = "Bourne shell" (Steven Bourne)
default on many (older) systems including Unix, non-POSIX

1970's: csh/tcsh = a "C like shell" (Bill Joy - BSD Unix)
MacOS, CentOS, extra install on Ubuntu/Debian, non-POSIX
tcsh fixes bugs in older csh but is incompatible with other shells
— why csh scripting is not recommended —
http://www.faqs.org/faqs/unix-faq/shell/csh-whynot
https://www.grymoire.com/Unix/CshTop10.txt

1983: ksh = "Korn shell" (David Korn, Bell Labs)
some compatibility with Bourne shell, mostly POSIX

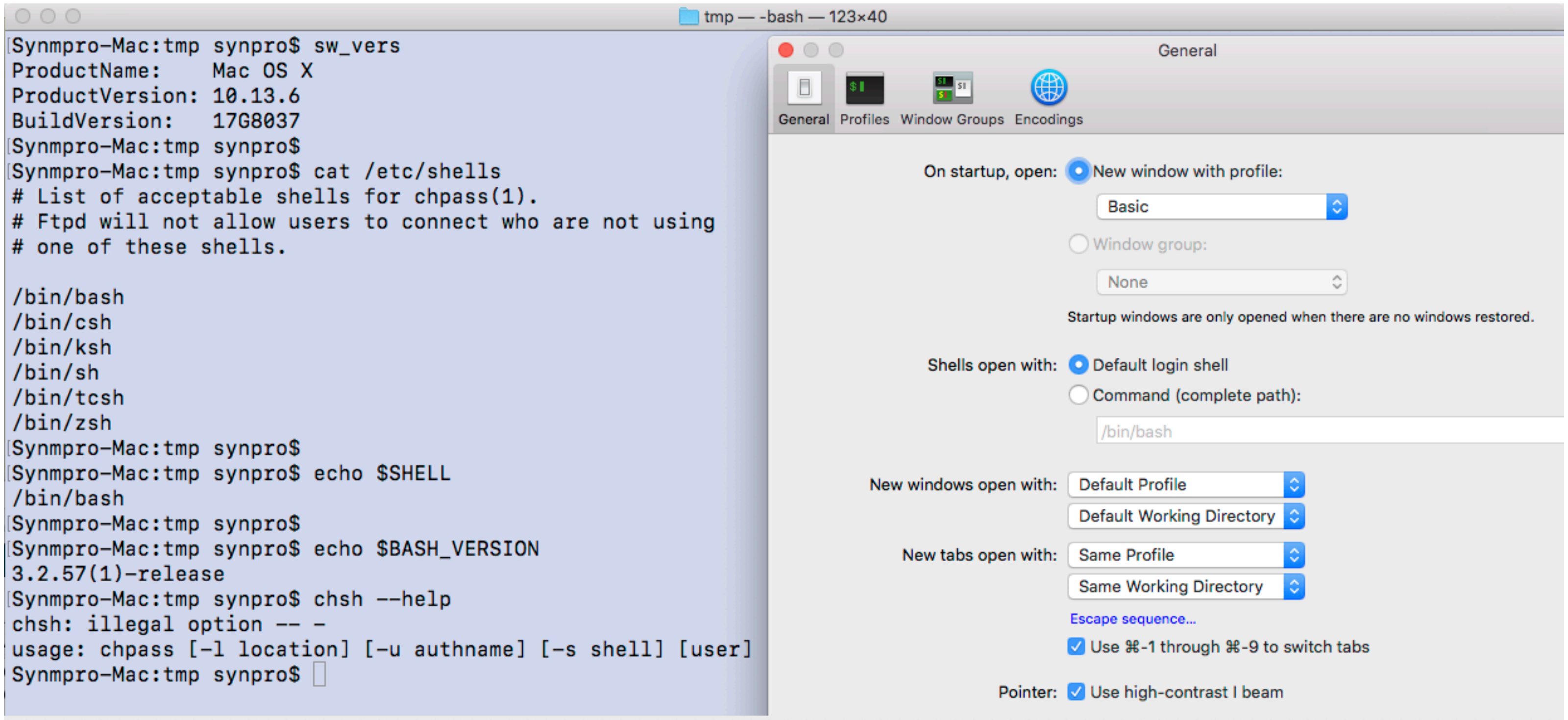
1989: bash = "Bourne-again shell" (GNU Software Project)
the default on MacOS (<= 10.14) & most Linux OS's like CentOS
bash has added features not available in Bourne shell - POSIX

late 1980's: ash = "almost shell" (Ken Almquist)
1997: dash = "Debian Almquist shell" (Herbert Xu)
both are lightweight variations of bash shell
ash for portable devices
dash on Ubuntu and Debian Linux - virtually POSIX

1990: zsh = alternative shell (Paul Falstad, Princeton Univ.)
soon to be the default in Mac OS 10.15 (Catalina)
features from bash, csh and ksh - ?? restrict to POSIX ??

POSIX = Portable Operating Systems Interface Standard (an API)
for software compatibility across OS's
(IEEE Computer Society standard)

TERMINAL - default shell and shell options



```
Synmpro-Mac:tmp synpro$ sw_vers
ProductName:    Mac OS X
ProductVersion: 10.13.6
BuildVersion:   17G8037
Synmpro-Mac:tmp synpro$
Synmpro-Mac:tmp synpro$ cat /etc/shells
# List of acceptable shells for chpass(1).
# Ftpd will not allow users to connect who are not using
# one of these shells.

/bin/bash
/bin/csh
/bin/ksh
/bin/sh
/bin/tcsh
/bin/zsh
Synmpro-Mac:tmp synpro$
Synmpro-Mac:tmp synpro$ echo $SHELL
/bin/bash
Synmpro-Mac:tmp synpro$
Synmpro-Mac:tmp synpro$ echo $BASH_VERSION
3.2.57(1)-release
Synmpro-Mac:tmp synpro$ chsh --help
chsh: illegal option -- -
usage: chpass [-l location] [-u authname] [-s shell] [user]
Synmpro-Mac:tmp synpro$
```

On startup, open: [✓ New window with profile:] Basic

Window group: None

Startup windows are only opened when there are no windows restored.

Shells open with: [✓ Default login shell] Command (complete path): /bin/bash

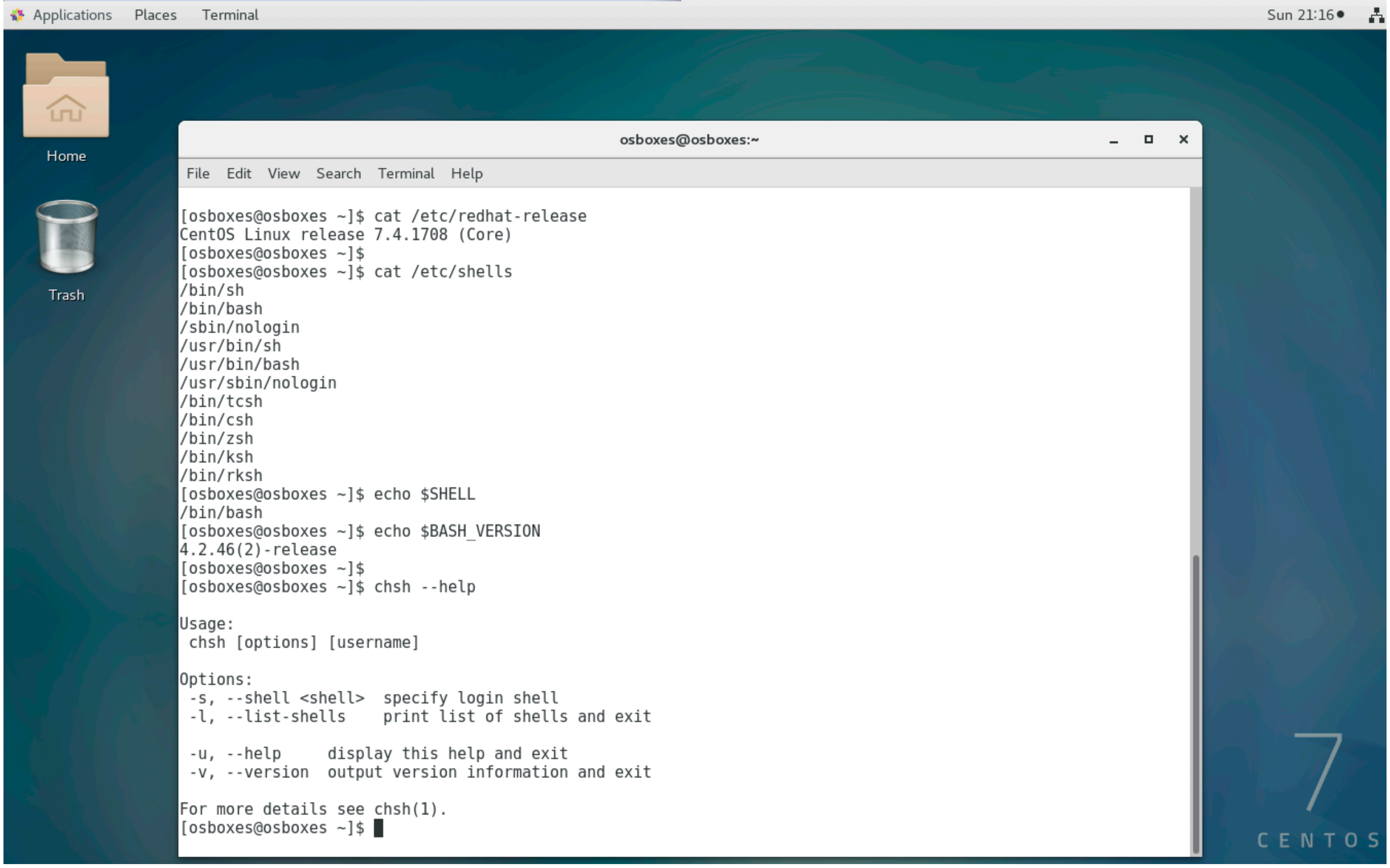
New windows open with: Default Profile — Default Working Directory

New tabs open with: Same Profile — Same Working Directory

Escape sequence...

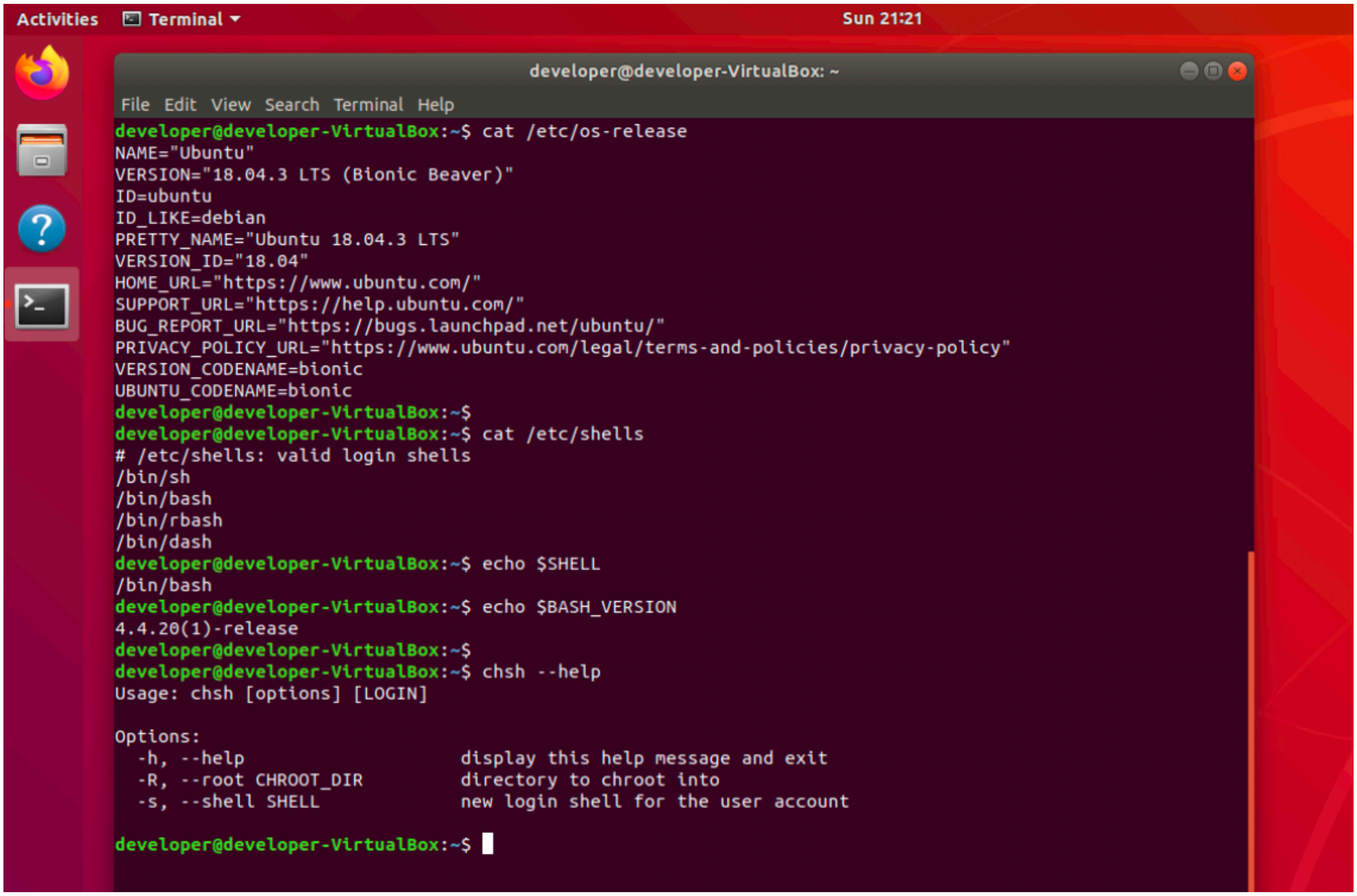
[✓] Use ⌘-1 through ⌘-9 to switch tabs

Pointer: [✓] Use high-contrast I beam



```
[osboxes@osboxes ~]$ cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/tcsh
/bin/csh
/bin/zsh
/bin/ksh
/bin/rksh
[osboxes@osboxes ~]$ echo $SHELL
/bin/bash
[osboxes@osboxes ~]$ echo $BASH_VERSION
4.2.46(2)-release
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ chsh --help

Usage:
 chsh [options] [username]

Options:
 -s, --shell <shell>  specify login shell
 -l, --list-shells    print list of shells and exit

 -u, --help     display this help and exit
 -v, --version  output version information and exit

For more details see chsh(1).
[osboxes@osboxes ~]$
```

TERMINAL - default shell and shell options

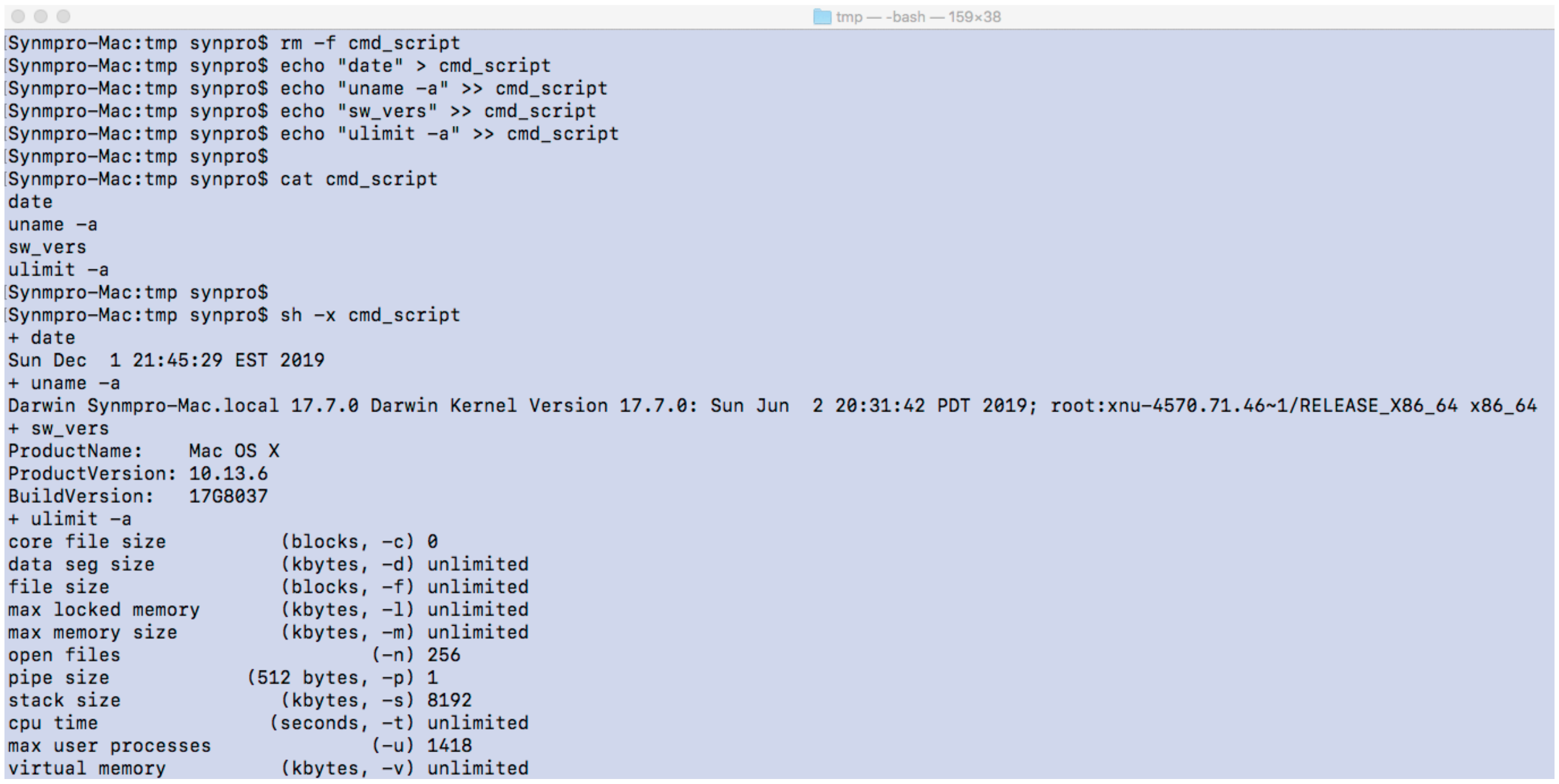

Activities
Terminal
Sun 21:21
developer@developer-VirtualBox: ~
File Edit View Search Terminal Help

```
developer@developer-VirtualBox:~$ cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
developer@developer-VirtualBox:~$
developer@developer-VirtualBox:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/bin/rbash
/bin/dash
developer@developer-VirtualBox:~$ echo $SHELL
/bin/bash
developer@developer-VirtualBox:~$ echo $BASH_VERSION
4.4.20(1)-release
developer@developer-VirtualBox:~$
developer@developer-VirtualBox:~$ chsh --help
Usage: chsh [options] [LOGIN]

Options:
  -h, --help                    display this help message and exit
  -R, --root CHROOT_DIR         directory to chroot into
  -s, --shell SHELL             new login shell for the user account

developer@developer-VirtualBox:~$ 
```

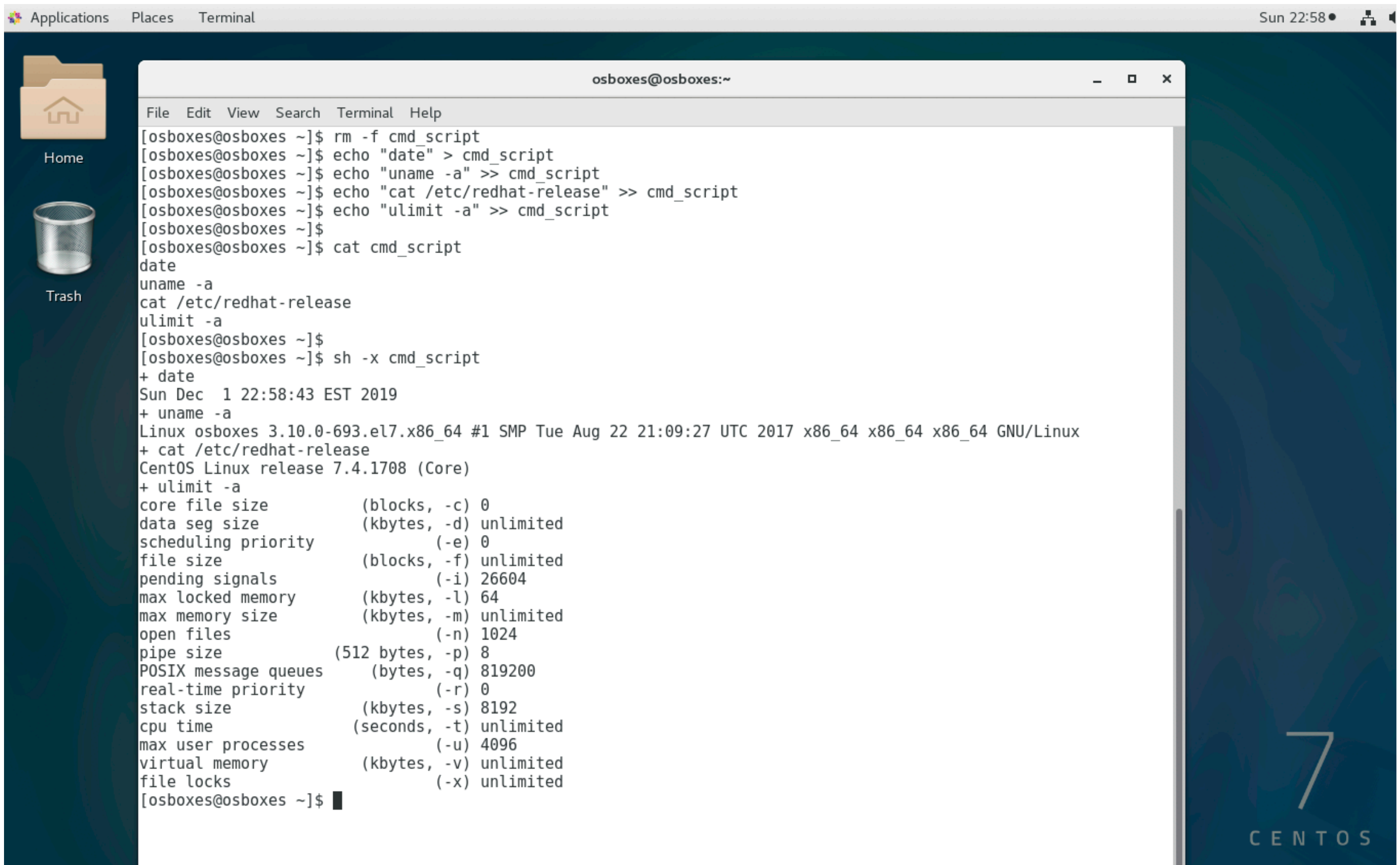
Available shells listed in /etc/shells depends upon what is installed on the machine

/bin/bash may not be the same version on all machines

SHELL - default limits & machine info (in a simple script)

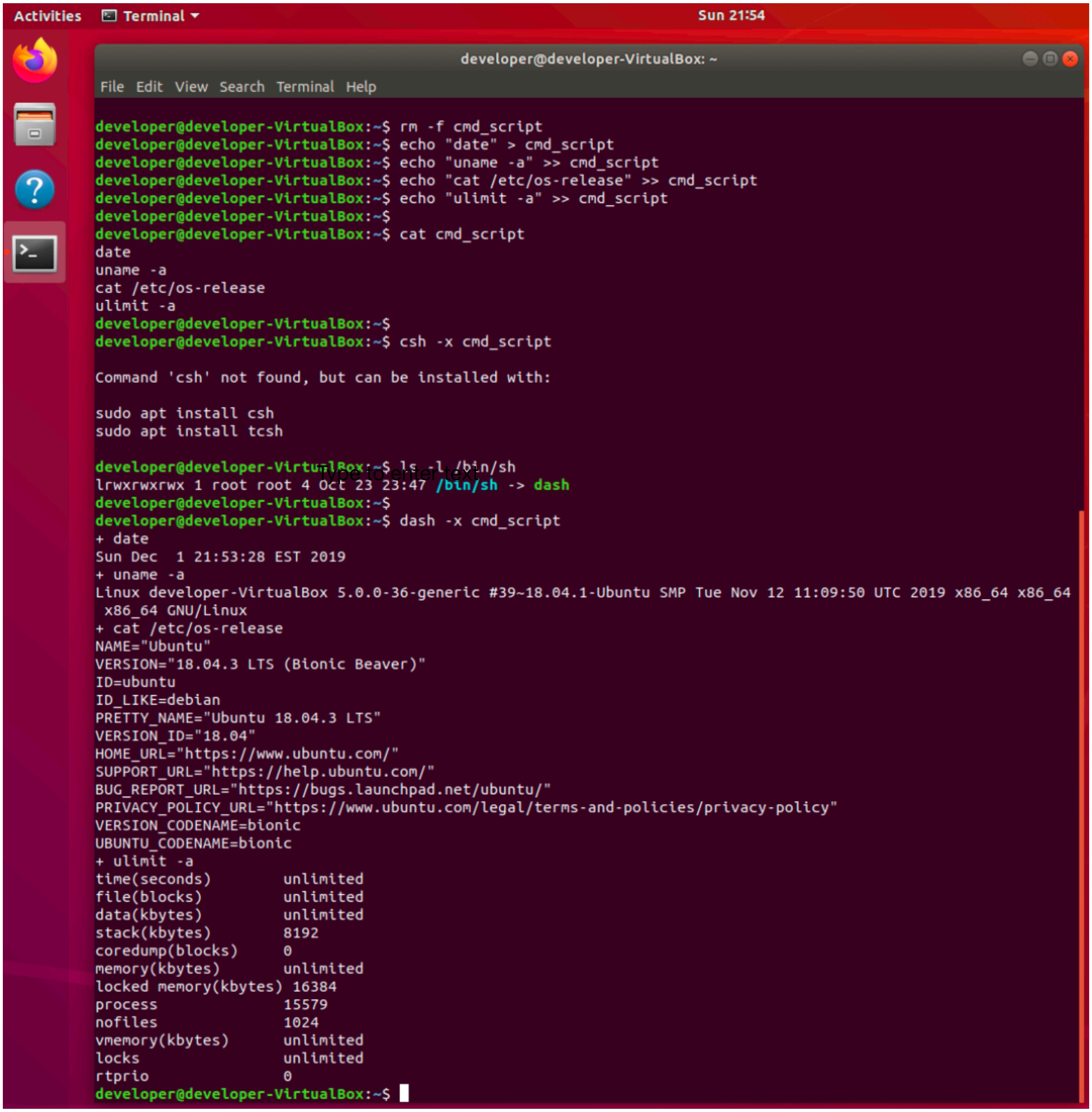


```
Synmpro-Mac:tmp synpro$ rm -f cmd_script
Synmpro-Mac:tmp synpro$ echo "date" > cmd_script
Synmpro-Mac:tmp synpro$ echo "uname -a" >> cmd_script
Synmpro-Mac:tmp synpro$ echo "sw_vers" >> cmd_script
Synmpro-Mac:tmp synpro$ echo "ulimit -a" >> cmd_script
Synmpro-Mac:tmp synpro$
Synmpro-Mac:tmp synpro$ cat cmd_script
date
uname -a
sw_vers
ulimit -a
Synmpro-Mac:tmp synpro$
Synmpro-Mac:tmp synpro$ sh -x cmd_script
+ date
Sun Dec  1 21:45:29 EST 2019
+ uname -a
Darwin Synmpro-Mac.local 17.7.0 Darwin Kernel Version 17.7.0: Sun Jun  2 20:31:42 PDT 2019; root:xnu-4570.71.46~1/RELEASE_X86_64 x86_64
+ sw_vers
ProductName:    Mac OS X
ProductVersion: 10.13.6
BuildVersion:   17G8037
+ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
max locked memory       (kbytes, -l) unlimited
max memory size         (kbytes, -m) unlimited
open files                      (-n) 256
pipe size            (512 bytes, -p) 1
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 1418
virtual memory          (kbytes, -v) unlimited
```



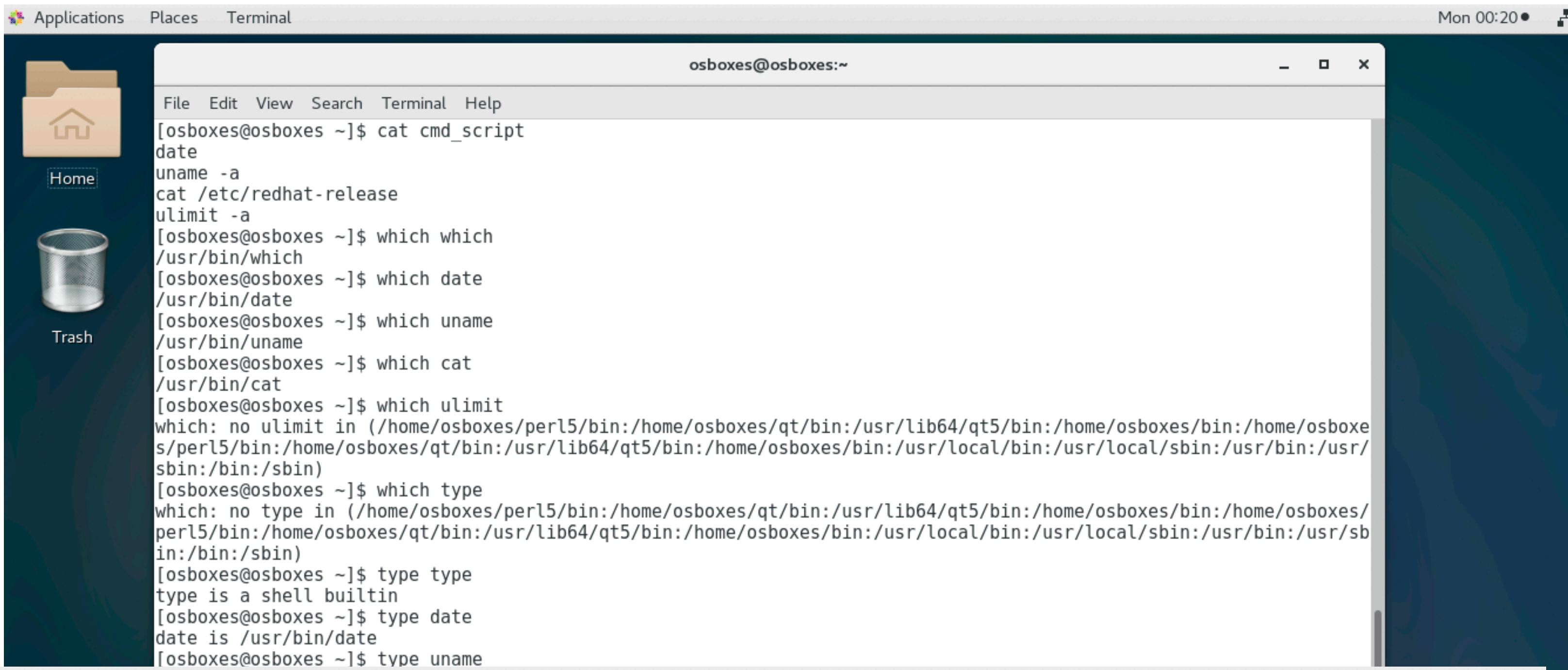
```
[osboxes@osboxes ~]$ rm -f cmd_script
[osboxes@osboxes ~]$ echo "date" > cmd_script
[osboxes@osboxes ~]$ echo "uname -a" >> cmd_script
[osboxes@osboxes ~]$ echo "cat /etc/redhat-release" >> cmd_script
[osboxes@osboxes ~]$ echo "ulimit -a" >> cmd_script
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ cat cmd_script
date
uname -a
cat /etc/redhat-release
ulimit -a
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ sh -x cmd_script
+ date
Sun Dec  1 22:58:43 EST 2019
+ uname -a
Linux osboxes 3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
+ cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
+ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 26604
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 4096
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
[osboxes@osboxes ~]$ █
```

SHELL - default limits & machine info (in a simple script)

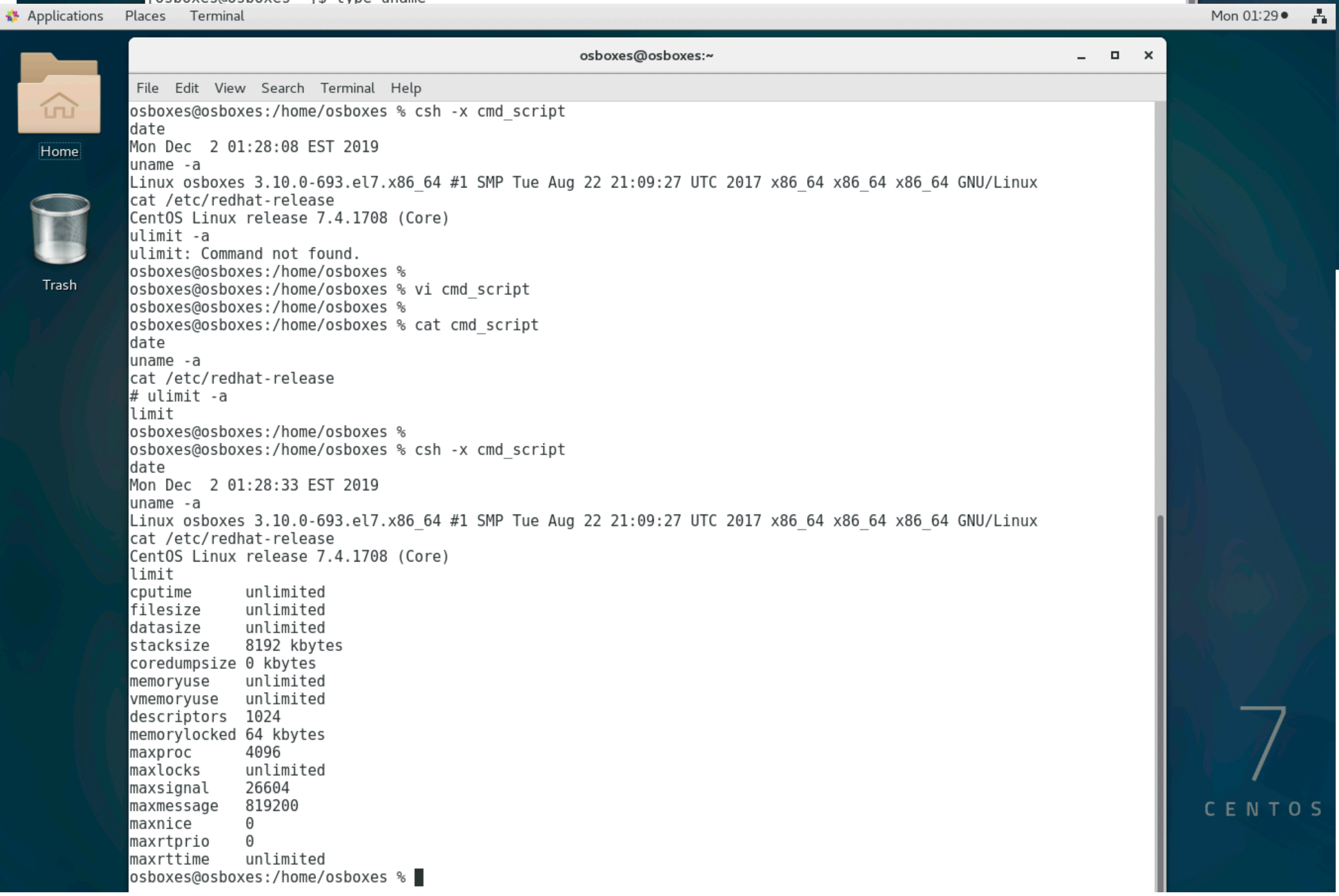

Activities
Terminal
Sun 21:54
developer@developer-VirtualBox: ~
File Edit View Search Terminal Help

```
developer@developer-VirtualBox:~$ rm -f cmd_script
developer@developer-VirtualBox:~$ echo "date" > cmd_script
developer@developer-VirtualBox:~$ echo "uname -a" >> cmd_script
developer@developer-VirtualBox:~$ echo "cat /etc/os-release" >> cmd_script
developer@developer-VirtualBox:~$ echo "ulimit -a" >> cmd_script
developer@developer-VirtualBox:~$
developer@developer-VirtualBox:~$ cat cmd_script
date
uname -a
cat /etc/os-release
ulimit -a
developer@developer-VirtualBox:~$
developer@developer-VirtualBox:~$ csh -x cmd_script

Command 'csh' not found, but can be installed with:

sudo apt install csh
sudo apt install tcsh

developer@developer-VirtualBox:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Oct 23 23:47 /bin/sh -> dash
developer@developer-VirtualBox:~$
developer@developer-VirtualBox:~$ dash -x cmd_script
+ date
Sun Dec  1 21:53:28 EST 2019
+ uname -a
Linux developer-VirtualBox 5.0.0-36-generic #39~18.04.1-Ubuntu SMP Tue Nov 12 11:09:50 UTC 2019 x86_64 x86_64
 x86_64 GNU/Linux
+ cat /etc/os-release
NAME="Ubuntu"
VERSION="18.04.3 LTS (Bionic Beaver)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 18.04.3 LTS"
VERSION_ID="18.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-policy"
VERSION_CODENAME=bionic
UBUNTU_CODENAME=bionic
+ ulimit -a
time(seconds)        unlimited
file(blocks)         unlimited
data(kbytes)         unlimited
stack(kbytes)        8192
coredump(blocks)     0
memory(kbytes)       unlimited
locked memory(kbytes) 16384
process              15579
nofiles              1024
vmemory(kbytes)      unlimited
locks                unlimited
rtprio               0
developer@developer-VirtualBox:~$
```

SHELL - where commands come from



```
[osboxes@osboxes ~]$ cat cmd_script
date
uname -a
cat /etc/redhat-release
ulimit -a
[osboxes@osboxes ~]$ which which
/usr/bin/which
[osboxes@osboxes ~]$ which date
/usr/bin/date
[osboxes@osboxes ~]$ which uname
/usr/bin/uname
[osboxes@osboxes ~]$ which cat
/usr/bin/cat
[osboxes@osboxes ~]$ which ulimit
which: no ulimit in (/home/osboxes/perl5/bin:/home/osboxes/qt/bin:/usr/lib64/qt5/bin:/home/osboxes/bin:/home/osboxe
s/perl5/bin:/home/osboxes/qt/bin:/usr/lib64/qt5/bin:/home/osboxes/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/
sbin:/bin:/sbin)
[osboxes@osboxes ~]$ which type
which: no type in (/home/osboxes/perl5/bin:/home/osboxes/qt/bin:/usr/lib64/qt5/bin:/home/osboxes/bin:/home/osboxes/
perl5/bin:/home/osboxes/qt/bin:/usr/lib64/qt5/bin:/home/osboxes/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sb
in:/bin:/sbin)
[osboxes@osboxes ~]$ type type
type is a shell builtin
[osboxes@osboxes ~]$ type date
date is /usr/bin/date
[osboxes@osboxes ~]$ type uname
```



```
osboxes@osboxes:/home/osboxes % csh -x cmd_script
date
Mon Dec  2 01:28:08 EST 2019
uname -a
Linux osboxes 3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
ulimit -a
ulimit: Command not found.
osboxes@osboxes:/home/osboxes %
osboxes@osboxes:/home/osboxes % vi cmd_script
osboxes@osboxes:/home/osboxes %
osboxes@osboxes:/home/osboxes % cat cmd_script
date
uname -a
cat /etc/redhat-release
# ulimit -a
limit
osboxes@osboxes:/home/osboxes %
osboxes@osboxes:/home/osboxes % csh -x cmd_script
date
Mon Dec  2 01:28:33 EST 2019
uname -a
Linux osboxes 3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux
cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
limit
cputime      unlimited
filesize     unlimited
datasize     unlimited
stacksize    8192 kbytes
coredumpsize 0 kbytes
memoryuse    unlimited
vmemoryuse   unlimited
descriptors  1024
memorylocked 64 kbytes
maxproc      4096
maxlocks     unlimited
maxsignal    26604
maxmessage   819200
maxnice      0
maxrtprio    0
maxrttime    unlimited
osboxes@osboxes:/home/osboxes %
```

SHELL - scripting basic terminal commands

Commands you run from the terminal should run inside a shell script.

Commands may be programs installed on the machine found thru the $PATH environment variable as set in the current shell

Commands may be "shell builtin" functions which are not the same in all shells, e.g., "ulimit" in sh/bash and "limit" in csh.

The simple shell script above (file cmd_script) contains 4 commands:
"date", "uname" and "cat" are installed programs found via $PATH
"ulimit" is a sh/bash builtin (except on the Mac)

Use the "which" program or the "type" shell builtin to see if a command is found in $PATH or if it is a shell builtin

csh/tcsh NOT INSTALLED BY DEFAULT on some linux distros, e.g., Ubuntu

/bin/sh may not be the same program on all linux distros !! e.g., on Ubuntu /bin/sh is actually the (light weight) dash shell

Any shell can run a file with generic commands found in $PATH, but this will no longer work once shell specific builtins or if, while, etc. statements are used

Explicitly exec a script in a different shell from the current shell by running it from the terminal just like any other command (add -x for debug), e.g.,

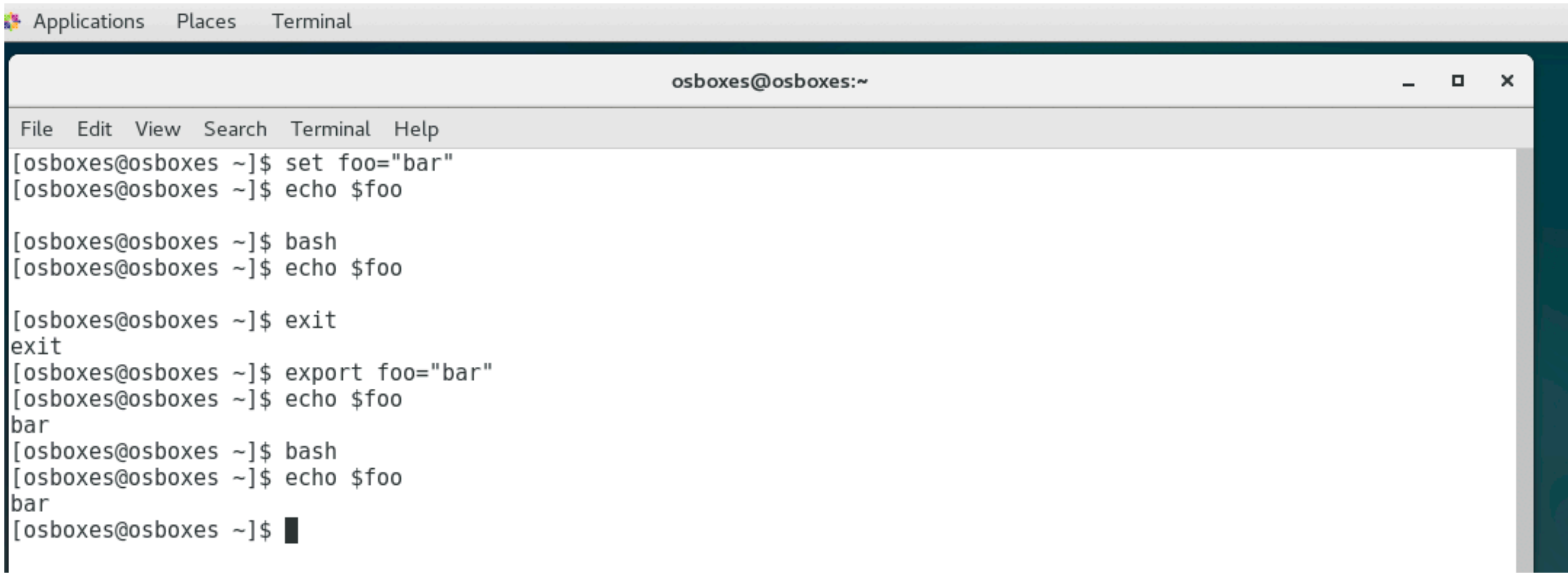
```
sh -x cmd_script
csh -x cmd_script
ksh -x cmd_script
```

Note that default (soft) shell limits can be different among machines:
Mac OS 10.13 open file limit = 256
Cent OS 7 open file limit = 1024
Ubuntu 18 open file limit = 1024

BTW - if you run a program that exceeds the current resource limits in your shell, then your program will fail, i.e. some scripts modify limit size to "unlimited"

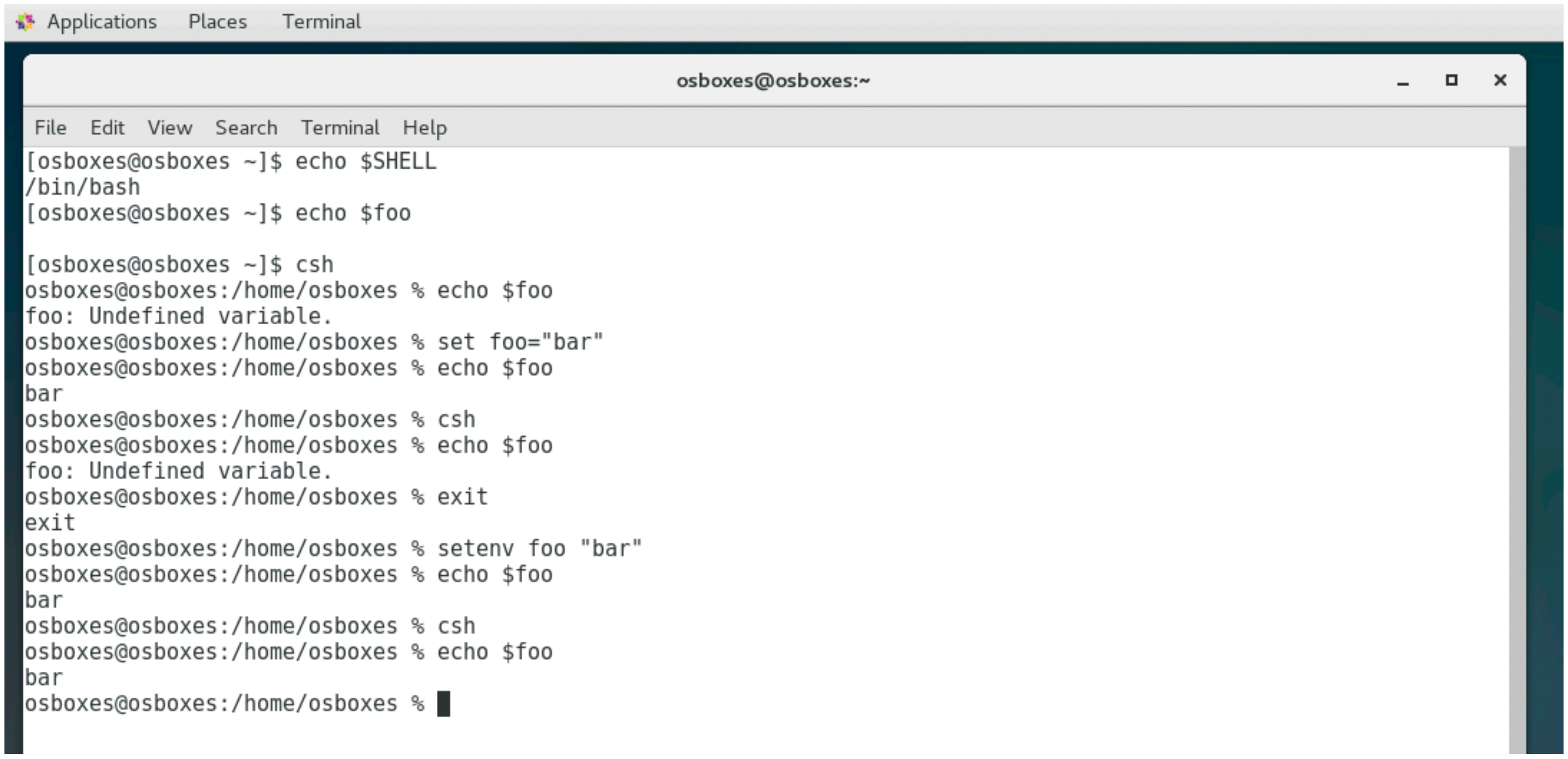
SHELL - environment variables

sh/bash: SET a variable to work only in the CURRENT PROCESS
sh/bash: EXPORT a variable to work in CURRENT and CHILD PROCESSES



```
[osboxes@osboxes ~]$ set foo="bar"
[osboxes@osboxes ~]$ echo $foo

[osboxes@osboxes ~]$ bash
[osboxes@osboxes ~]$ echo $foo

[osboxes@osboxes ~]$ exit
exit
[osboxes@osboxes ~]$ export foo="bar"
[osboxes@osboxes ~]$ echo $foo
bar
[osboxes@osboxes ~]$ bash
[osboxes@osboxes ~]$ echo $foo
bar
[osboxes@osboxes ~]$ 
```

csh/tcsh: SET a variable to work only in the CURRENT PROCESS
csh/tcsh: SETENV a variable to work in CURRENT and CHILD PROCESSES



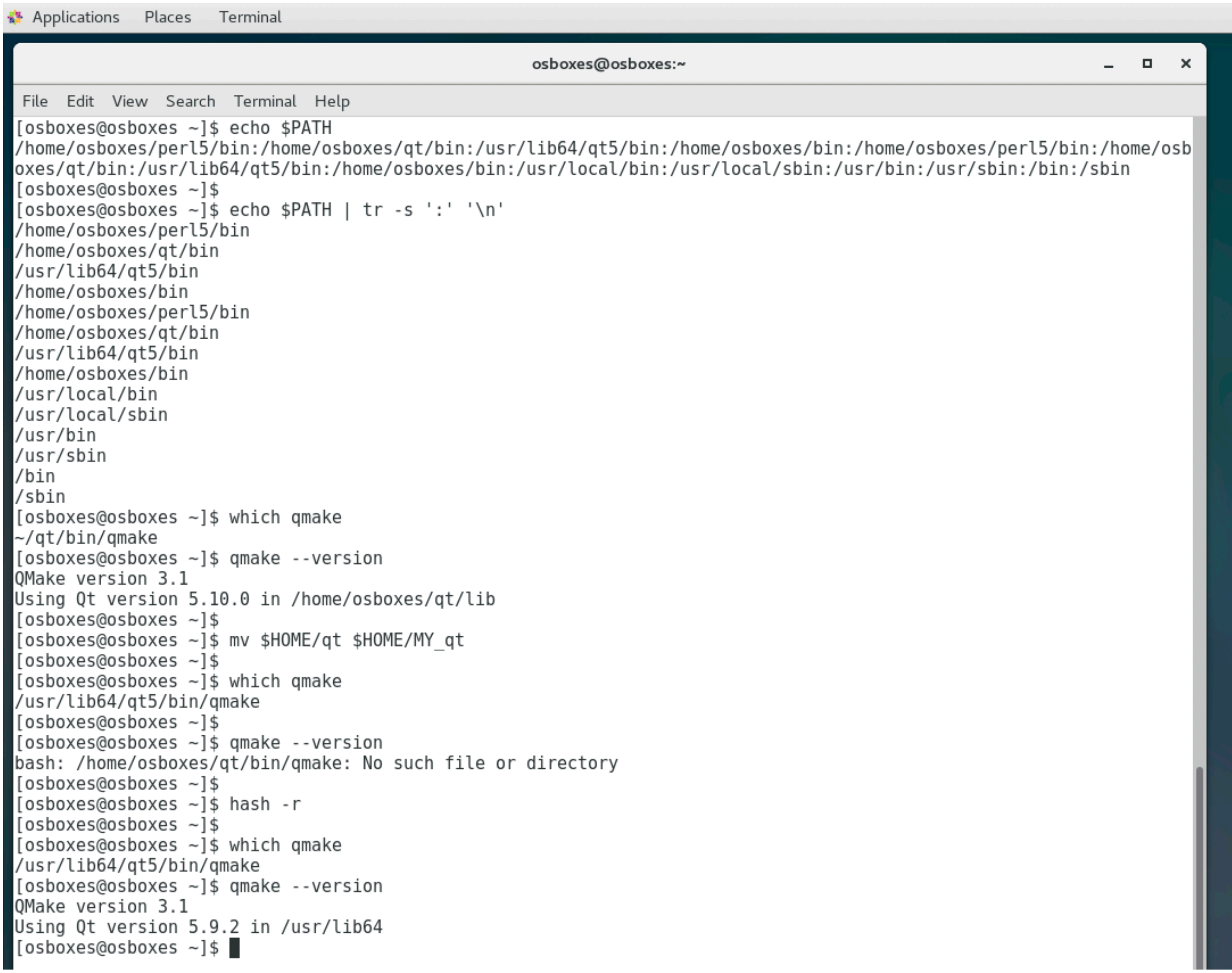
```
[osboxes@osboxes ~]$ echo $SHELL
/bin/bash
[osboxes@osboxes ~]$ echo $foo

[osboxes@osboxes ~]$ csh
osboxes@osboxes:/home/osboxes % echo $foo
foo: Undefined variable.
osboxes@osboxes:/home/osboxes % set foo="bar"
osboxes@osboxes:/home/osboxes % echo $foo
bar
osboxes@osboxes:/home/osboxes % csh
osboxes@osboxes:/home/osboxes % echo $foo
foo: Undefined variable.
osboxes@osboxes:/home/osboxes % exit
exit
osboxes@osboxes:/home/osboxes % setenv foo "bar"
osboxes@osboxes:/home/osboxes % echo $foo
bar
osboxes@osboxes:/home/osboxes % csh
osboxes@osboxes:/home/osboxes % echo $foo
bar
osboxes@osboxes:/home/osboxes % 
```

SHELL - environment variables

PATH is critical - the ordering of PATH entries can make the difference between your programs running (or your source code compiling) correctly

The first matching entry in PATH is used for a binary - if changing PATH or binary locations from the command line use the shell re-hash function

bash: hash -r
csh/tcsh: rehash


Applications Places Terminal

osboxes@osboxes:~

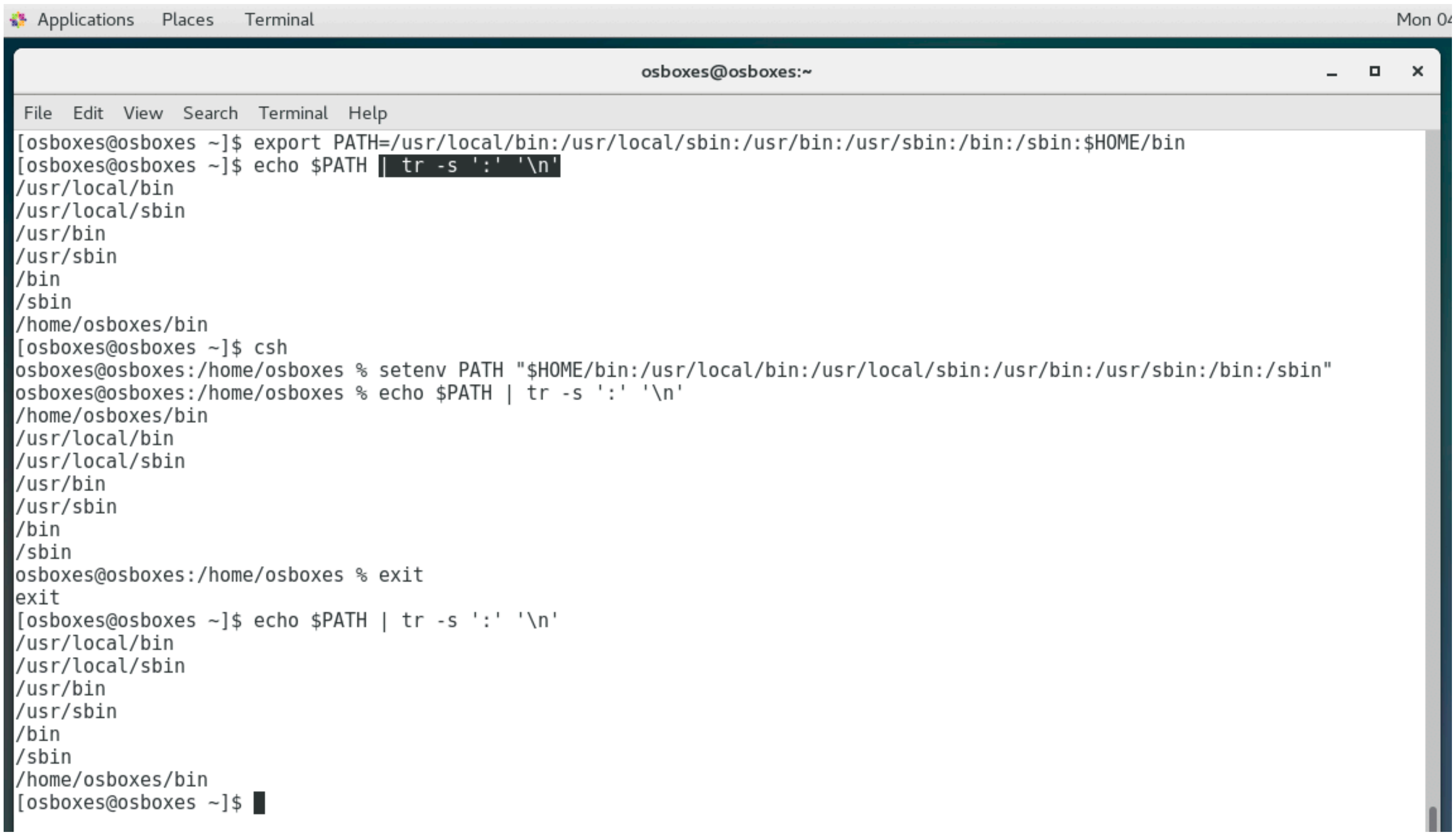
File Edit View Search Terminal Help

```
[osboxes@osboxes ~]$ echo $PATH
/home/osboxes/perl5/bin:/home/osboxes/qt/bin:/usr/lib64/qt5/bin:/home/osboxes/bin:/home/osboxes/perl5/bin:/home/osb
oxes/qt/bin:/usr/lib64/qt5/bin:/home/osboxes/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ echo $PATH | tr -s ':' '\n'
/home/osboxes/perl5/bin
/home/osboxes/qt/bin
/usr/lib64/qt5/bin
/home/osboxes/bin
/home/osboxes/perl5/bin
/home/osboxes/qt/bin
/usr/lib64/qt5/bin
/home/osboxes/bin
/usr/local/bin
/usr/local/sbin
/usr/bin
/usr/sbin
/bin
/sbin
[osboxes@osboxes ~]$ which qmake
~/qt/bin/qmake
[osboxes@osboxes ~]$ qmake --version
QMake version 3.1
Using Qt version 5.10.0 in /home/osboxes/qt/lib
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ mv $HOME/qt $HOME/MY_qt
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ which qmake
/usr/lib64/qt5/bin/qmake
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ qmake --version
bash: /home/osboxes/qt/bin/qmake: No such file or directory
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ hash -r
[osboxes@osboxes ~]$
[osboxes@osboxes ~]$ which qmake
/usr/lib64/qt5/bin/qmake
[osboxes@osboxes ~]$ qmake --version
QMake version 3.1
Using Qt version 5.9.2 in /usr/lib64
[osboxes@osboxes ~]$ █
```

SHELL - environment variables

Order $HOME/bin to be first at tail of PATH
and then at head of PATH

sh/bash: export PATH=/path1:/path2:/path3…
csh/tcsh: setenv PATH “/path1:/path2:/path3…”



```
[osboxes@osboxes ~]$ export PATH=/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:$HOME/bin
[osboxes@osboxes ~]$ echo $PATH | tr -s ':' '\n'
/usr/local/bin
/usr/local/sbin
/usr/bin
/usr/sbin
/bin
/sbin
/home/osboxes/bin
[osboxes@osboxes ~]$ csh
osboxes@osboxes:/home/osboxes % setenv PATH "$HOME/bin:/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin"
osboxes@osboxes:/home/osboxes % echo $PATH | tr -s ':' '\n'
/home/osboxes/bin
/usr/local/bin
/usr/local/sbin
/usr/bin
/usr/sbin
/bin
/sbin
osboxes@osboxes:/home/osboxes % exit
exit
[osboxes@osboxes ~]$ echo $PATH | tr -s ':' '\n'
/usr/local/bin
/usr/local/sbin
/usr/bin
/usr/sbin
/bin
/sbin
/home/osboxes/bin
[osboxes@osboxes ~]$
```

SHELL - customization with shell init files

Automatically set environment variables in your terminal windows (after you login)

Use sh/bash or csh/tcsh specific syntax/grammar
if statements, export, setenv, set, etc. shell commands

sh/bash: $HOME/.bash_profile or $HOME/.bashrc
csh/tcsh: $HOME/.cshrc or $HOME/.tcshrc

```
# in .cshrc add local freesurfer binaries at tail of path, add vars for mne and matlab

setenv FREESURFER_HOME /usr/local/freesurfer/stable6
set path = ( $path $FREESURFER_HOME/bin)

setenv MNE_ROOT /usr/pubsw/packages/mne/stable
setenv MATLAB_ROOT /usr/pubsw/packages/matlab/current

# in .cshrc use anaconda python only if on MacOS - test shell builtin OSTYPE variable

# set os = `uname -s | tr -s '[A-Z]' '[a-z]'`
if ($OSTYPE == "darwin") then
  set conda_bin='/Volumes/extra/anaconda3/bin'
  alias condapython ${conda_bin}/python
  set path = ($path $conda_bin)
endif
```

```
# in .bashrc set freesurfer environment with tutorial data on local storage only if
# on a specific machine - test the output of the hostname command

export FREESURFER_HOME=/usr/local/freesurfer/stable6
export PATH=$PATH:$FREESURFER_HOME/bin

HOSTNAME=`hostname`
if [ "$HOSTNAME" == "tanha" ] ; then
  export TUTORIAL_DATA=/local_mount/space/tanha/1/users/rc221/tutorial_data
fi
```

SHELL - customization with shell init files

Use aliases to define shortcuts from the terminal command line, e.g. the alias “make” will run the latest GNU make program

sh/bash: alias=<alias name> ‘<cmd>’
csh/tcsh: alias <alias name> “<cmd>”

```
# only create alias in .cshrc to use this binary if it exists

if ($OSTYPE == "darwin") then
      if ( -f /usr/local/bin/gmake ) then
      # use my local build of latest GNU make rev 4.2 when I type “make”
      alias make /usr/local/bin/gmake
   endif
endif

# in .cshrc redefine the ls command via an alias to show colors etc., for
# directories, different file types, …

if ($OSTYPE == "darwin") then
   alias ls ls -FG
else if ($OSTYPE == "linux") then
   alias ls ls -F --color
endif

# in .cshrc redefine prompt and cd command to display userid, host, current
# working directory, e.g., useful to copy and paste into scp like commands

set prompt="`whoami`@${name}:$cwd % "
alias setprompt 'set prompt="`whoami`@${name}:${cwd} % "'
alias cd 'chdir \!* && setprompt’

# GIT - what has not been pushed
alias gitp git diff --stat --cached origin
# GIT - list modified
alias glm git ls-files --modified
# GIT - add modified
alias gam git ls-files --modified | xargs git add
# GIT - prettier log output
alias gith git log --oneline --decorate --graph --all
alias gitp git log —pretty=oneline
```

SHELL - customization with shell init files

Display a SINGLE alias by typing “alias <cmd>” in terminal

Display ALL ALIASES by typing “alias” in terminal

```
$ csh
user@methos:/Users/user % echo $OSTYPE
darwin

user@methos:/Users/user % alias ls
ls -FG
user@methos:/Users/user % alias
cd      chdir !* && setprompt
cmake_clean        (git clean -d -f -x)
condapython        /Volumes/extra/anaconda3/bin/python
gdiff   /usr/local/bin/diff
git_show_commits         (git log origin..HEAD)
gitc    git log --date=short --pretty=format:%x0a%ci%x09%h%x09%\<\(15,trunc\)%an%x09%
\<\(55,trunc\)%s
gitcl   (git show --name-only)
gitdevtree    (git ls-tree -r dev --name-only)
gitguidiff      (git difftool -t tkdiff)
gith    (git log --oneline --decorate --graph --all)
gitls   git ls-files | xargs git log --pretty=format:%h%x09%an%x09%ad%x09%s --
since=2017-07-01 --name-only
gitp    (git log --pretty=oneline)
gitts   (git rev-list -n 1)
glm     (git ls-files --modified)
ll      ls -l
lr      ls -lR
ls      ls -FG
lt      ls -lt
make /usr/local/bin/gmake
setprompt    set prompt="`whoami`@${name}:${cwd} % "
st      cd /Volumes/extra/git/sbgrid/software-tests
tree    find . -print | sed -e 's;[^/]*/;|____;g;s;____|; |;g'
```

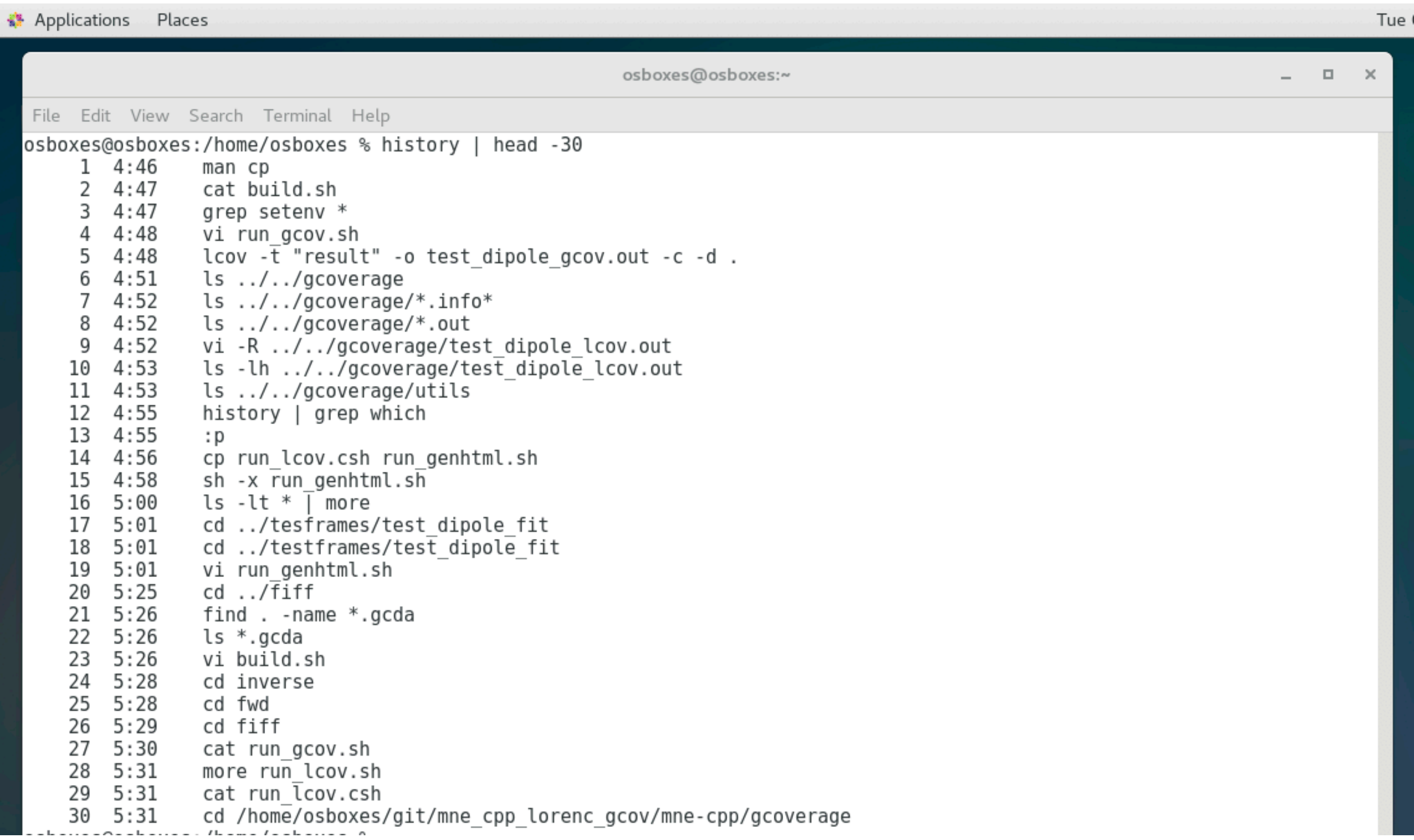
SHELL - set, save and list command history

Default history file is $HOME/.history
(change with “set histfile=<file>”)

History count in the active terminal window
set history=2000

History to be saved after logout
set savehist= 2000

Works in sh/bash and csh/tcsh shell init file



```
osboxes@osboxes:/home/osboxes % history | head -30
     1  4:46    man cp
     2  4:47    cat build.sh
     3  4:47    grep setenv *
     4  4:48    vi run_gcov.sh
     5  4:48    lcov -t "result" -o test_dipole_gcov.out -c -d .
     6  4:51    ls ../../gcoverage
     7  4:52    ls ../../gcoverage/*.info*
     8  4:52    ls ../../gcoverage/*.out
     9  4:52    vi -R ../../gcoverage/test_dipole_lcov.out
    10  4:53    ls -lh ../../gcoverage/test_dipole_lcov.out
    11  4:53    ls ../../gcoverage/utils
    12  4:55    history | grep which
    13  4:55    :p
    14  4:56    cp run_lcov.csh run_genhtml.sh
    15  4:58    sh -x run_genhtml.sh
    16  5:00    ls -lt * | more
    17  5:01    cd ../tesframes/test_dipole_fit
    18  5:01    cd ../testframes/test_dipole_fit
    19  5:01    vi run_genhtml.sh
    20  5:25    cd ../fiff
    21  5:26    find . -name *.gcda
    22  5:26    ls *.gcda
    23  5:26    vi build.sh
    24  5:28    cd inverse
    25  5:28    cd fwd
    26  5:29    cd fiff
    27  5:30    cat run_gcov.sh
    28  5:31    more run_lcov.sh
    29  5:31    cat run_lcov.csh
    30  5:31    cd /home/osboxes/git/mne_cpp_lorenc_gcov/mne-cpp/gcoverage
```

What are all the commands I typed containing “cmake” or “make” ?

$ **history | grep "[cm]ake" | more**

```
   22  CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9  CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9  FC=/
usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gfortran-4.9  /usr/local/bin/cmake -
DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DTIFF_LIB=/usr/local/Cellar/libtiff/4.0.9_5/lib/libtiff.dylib  -
DTIFF_INCLUDE_DIR=/usr/local/Cellar/libtiff/4.0.9_5/include -DXm_LIB=/usr/lib/libxml2.dylib -
DXm_INCLUDE_DIR=/usr/include/libxml2/libxml -DCMAKE_CXX_FLAGS:STRING="${CMAKE_CXX_FLAGS} -Wno-
unused-but-set-variable" -DUSE_MATLAB:BOOL=OFF -DMACOSX_RPATH="@loader_path/../lib/
libquicktime.dylib" ../mne-c
   23  make clean
   24  make -j1 2>&1 | tee -a make.log.1
   25  make clean
   35  cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DUSE_MATLAB:BOOL=OFF -DUSE_X11:BOOL=ON -
DHOMEBREW:BOOL=ON ../mne-c
   36  make -j1 2>&1 | tee -a make.log.1
   37  make -j1 install 2>&1 | tee -a install.log.1
   54  CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9  CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9  FC=/
usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gfortran-4.9  /usr/local/bin/cmake -
DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DTIFF_LIB=/usr/local/Cellar/libtiff/4.0.9_5/lib/libtiff.dylib  -
DTIFF_INCLUDE_DIR=/usr/local/Cellar/libtiff/4.0.9_5/include -DXm_LIB=/usr/lib/libxml2.dylib -
DXm_INCLUDE_DIR=/usr/include/libxml2/libxml -DCMAKE_CXX_FLAGS:STRING="${CMAKE_CXX_FLAGS} -Wno-
unused-but-set-variable" -DUSE_MATLAB:BOOL=OFF -DMACOSX_RPATH="@loader_path/../lib/
libquicktime.dylib" ../mne-c
   56  cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DUSE_MATLAB:BOOL=OFF -DUSE_X11:BOOL=ON -
DHOMEBREW:BOOL=ON ../mne-c
   72  vi ./dev-tools/cmake/FindBISON.cmake
   88  vi ./dev-tools/cmake/FindBISON.cmake ./dev-tools/cmake/FindFLEX.cmake ./dev-tools/cmake/
FindSubversion.cmake ./MNE/mne_analyze/mne_analyze.h ./MNE/mne_browse_raw/mne_browse_raw.h ./MNE/
mne_scripts/mne_setup ./MNE/mne_scripts/mne_setup_sh ./MNE/mne_simu/FindBISON.cmake ./MNE-libs/mne/
mne_add_geometry_info.c ./MNE-libs/mne/mne_forward_util.c ./MNE-libs/mne/mne_patches.c ./MNE-libs/mne/
mne_source_space.c ./MNE-libs/mne/mne_types.h ./other-libs/plotutils-2.5.1/libxmi/mkinstalldirs
```

$ **history > $HOME/h.build_try_1**

Save your command history periodically to a uniquely named file

Saved history useful if you are working on a difficult debugging or build session and can’t remember every command (or made mistakes with commands)

Print your history, review it, pick out the commands that worked, reproduce the results and turn it into a shell script!

SHELL - history - recall and run a specific command

```
$ history | grep cmake

   22  CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9  CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9  FC=/
usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gfortran-4.9  /usr/local/bin/cmake -
DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DTIFF_LIB=/usr/local/Cellar/libtiff/4.0.9_5/lib/libtiff.dylib  -
DTIFF_INCLUDE_DIR=/usr/local/Cellar/libtiff/4.0.9_5/include -DXm_LIB=/usr/lib/libxml2.dylib -
DXm_INCLUDE_DIR=/usr/include/libxml2/libxml -DCMAKE_CXX_FLAGS:STRING="${CMAKE_CXX_FLAGS} -Wno-
unused-but-set-variable" -DUSE_MATLAB:BOOL=OFF -DMACOSX_RPATH="@loader_path/../lib/
libquicktime.dylib" ../mne-c
   35  cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DUSE_MATLAB:BOOL=OFF -DUSE_X11:BOOL=ON -
DHOMEBREW:BOOL=ON ../mne-c
   54  CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9  CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9  FC=/
usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gfortran-4.9  /usr/local/bin/cmake -
DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DTIFF_LIB=/usr/local/Cellar/libtiff/4.0.9_5/lib/libtiff.dylib  -
DTIFF_INCLUDE_DIR=/usr/local/Cellar/libtiff/4.0.9_5/include -DXm_LIB=/usr/lib/libxml2.dylib -
DXm_INCLUDE_DIR=/usr/include/libxml2/libxml -DCMAKE_CXX_FLAGS:STRING="${CMAKE_CXX_FLAGS} -Wno-
unused-but-set-variable" -DUSE_MATLAB:BOOL=OFF -DMACOSX_RPATH="@loader_path/../lib/
libquicktime.dylib" ../mne-c
   56  cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DUSE_MATLAB:BOOL=OFF -DUSE_X11:BOOL=ON -
DHOMEBREW:BOOL=ON ../mne-c
   72  vi ./dev-tools/cmake/FindBISON.cmake
   88  vi ./dev-tools/cmake/FindBISON.cmake ./dev-tools/cmake/FindFLEX.cmake ./dev-tools/cmake/
FindSubversion.cmake ./MNE/mne_analyze/mne_analyze.h ./MNE/mne_browse_raw/mne_browse_raw.h ./MNE/
mne_scripts/mne_setup ./MNE/mne_scripts/mne_setup_sh ./MNE/mne_simu/FindBISON.cmake ./MNE-libs/mne/
mne_add_geometry_info.c ./MNE-libs/mne/mne_forward_util.c ./MNE-libs/mne/mne_patches.c ./MNE-libs/mne/
mne_source_space.c ./MNE-libs/mne/mne_types.h ./other-libs/plotutils-2.5.1/libxmi/mkinstalldirs
  138  git add MNE-libs/mne/mne_add_geometry_info.c MNE-libs/mne/mne_forward_util.c MNE-libs/mne/
mne_patches.c MNE-libs/mne/mne_source_space.c MNE-libs/mne/mne_types.h MNE/mne_analyze/
mne_analyze.h MNE/mne_browse_raw/mne_browse_raw.h MNE/mne_scripts/mne_setup MNE/mne_scripts/
mne_setup_sh MNE/mne_simu/FindBISON.cmake dev-tools/cmake/FindBISON.cmake dev-tools/cmake/
FindFLEX.cmake dev-tools/cmake/FindSubversion.cmake other-libs/plotutils-2.5.1/libxmi/mkinstalldirs
  511  cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/
mne_git_fork_alex_rob_homebrew_branch/install -DUSE_MATLAB:BOOL=OFF -DUSE_X11:BOOL=ON -
DHOMEBREW:BOOL=ON ../mne-c
$
$ !54:p   <—-  print command 54 which puts in at the top of the command line buffer
CC=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/gcc-4.9  CXX=/usr/local/Cellar/gcc@4.9/4.9.4_1/bin/g++-4.9  FC=/usr/
local/Cellar/gcc@4.9/4.9.4_1/bin/gfortran-4.9  /usr/local/bin/cmake -DCMAKE_VERBOSE_MAKEFILE:BOOL=ON -
DCMAKE_INSTALL_PREFIX=/Volumes/hd-3/mne_git_fork_alex_rob_homebrew_branch/install -DTIFF_LIB=/usr/
local/Cellar/libtiff/4.0.9_5/lib/libtiff.dylib  -DTIFF_INCLUDE_DIR=/usr/local/Cellar/libtiff/4.0.9_5/include -DXm_LIB=/
usr/lib/libxml2.dylib -DXm_INCLUDE_DIR=/usr/include/libxml2/libxml -DCMAKE_CXX_FLAGS:STRING="$
{CMAKE_CXX_FLAGS} -Wno-unused-but-set-variable" -DUSE_MATLAB:BOOL=OFF -
DMACOSX_RPATH="@loader_path/../lib/libquicktime.dylib" ../mne-c
Synmpro-Mac:~ synpro$ !!   <—- execute the last command in the buffer (now command 54) or
                                press up arrow key once to navigate thru history
Synmpro-Mac:~ synpro$ !54  <—- or could have exec'd command 54 explicitly by number
```

SHELL - multiple shells - compare settings

My bash and csh init files have different settings, e.g., PATH, aliases, etc.

What happens if I am in bash and then exec csh from the bash shell?

csh inherits no settings from bash ?
csh inherits all settings from bash ?
csh overwrites common settings from bash ?

```
$ cd /tmp
$ bash
user@pro-Mac:/tmp> echo $SHELL    <—- bash prompt may be $ or > sign
/bin/bash
user@pro-Mac:/tmp> csh
user@pro-Mac:/tmp % echo $SHELL. <—- csh prompt is usually %
/bin/csh
user@pro-Mac:/tmp %
```

Use the env command to examine and compare shell environments

Output a sorted list of environment variables to different files

sh/bash: env | sort > /tmp/env.1
csh/tcsh: env | sort > /tmp/env.2

```
user@pro-Mac:/tmp> env | sort > /tmp/env.1   <—- save BASH environment
user@pro-Mac:/tmp> env | sort
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.0fzfF6hRJv/Render
DISPLAY=/private/tmp/com.apple.launchd.1zjX3wleBy/org.macosforge.xquartz:0
HISTCONTROL=ignorespace
HOME=/Users/user
LANG=en_US.UTF-8
LOGNAME=user
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin:/usr/local/bin
PS1=\[\e]0;\u@\h: \w\a\]\[\033[01;31m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]>
PWD=/tmp
SECURITYSESSIONID=186a7
SHELL=/bin/bash
SHLVL=2
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.9mkFwYeBHy/Listeners
TERM=xterm-256color
TERM_PROGRAM=Apple_Terminal
TERM_PROGRAM_VERSION=404.1
TERM_SESSION_ID=33BF57FF-E447-4044-99EF-2050A638642C
TMPDIR=/var/folders/6q/ty58ddr52h5ggfd7c96x25_c0000gn/T/
USER=user
XPC_FLAGS=0x0
XPC_SERVICE_NAME=0
_=/usr/bin/env
user@pro-Mac:/tmp> csh
user@pro-Mac:/tmp % env | sort > /tmp/env.2.  <—- save CSH environment
user@pro-Mac:/tmp % env | sort
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.0fzfF6hRJv/Render
DISPLAY=/private/tmp/com.apple.launchd.1zjX3wleBy/org.macosforge.xquartz:0
FOO=bar
GITHUB_USER=buildqa
GROUP=staff
HISTCONTROL=ignorespace
HOME=/Users/user
HOST=Synmpro-Mac.local
HOSTTYPE=unknown
LANG=en_US.UTF-8
LOGNAME=buildqa
MACHTYPE=x86_64
OSTYPE=darwin
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin:/usr/local/bin:/Users/user/bin
PS1=\[\e]0;\u@\h: \w\a\]\[\033[01;31m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]>
PWD=/tmp
SECURITYSESSIONID=186a7
SHELL=/bin/csh
SHLVL=3
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.9mkFwYeBHy/Listeners
TERM=xterm-256color
TERM_PROGRAM=Apple_Terminal
TERM_PROGRAM_VERSION=404.1
TERM_SESSION_ID=33BF57FF-E447-4044-99EF-2050A638642C
TMPDIR=/var/folders/6q/ty58ddr52h5ggfd7c96x25_c0000gn/T/
USER=user
VENDOR=apple
XPC_FLAGS=0x0
```

SHELL - multiple shell environments - compare settings

```
user@pro-MAC:/tmp % diff /tmp/env.1 /tmp/env.2
2a3,5
> FOO=bar
> GITHUB_USER=buildqa
> GROUP=staff
4a8,9
> HOST=pro-Mac.local
> HOSTTYPE=unknown
6,7c11,14
< LOGNAME=user
< PATH=/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin:/usr/local/bin
---
> LOGNAME=buildqa
> MACHTYPE=x86_64
> OSTYPE=darwin
> PATH=/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin:/usr/local/bin:/Users/user/bin
11,12c18,19
< SHELL=/bin/bash
< SHLVL=2
---
> SHELL=/bin/csh
> SHLVL=3
19a27
> VENDOR=apple
22c30
< _=/usr/bin/env
---
> _=/bin/csh
```

The csh gets what was exported into the bash shell, and the csh init files will subsequently set its own and modify any common variables, e.g., PATH

Watch out for differences in *PATH* variables between shell environments. Differences in finding binaries or library search paths can cause programs to run/fail/compile differently

Linux: PATH, LD_LIBRARY_PATH, PYTHONPATH, etc…

Mac: PATH, DYLD_LIBRARY_PATH, DYLD_FALLBACK_LIBRARY_PATH, PYTHONPATH, etc…

SHELL - multiple shell environments - suspend and return

```
user@pro-Mac:/tmp> echo $SHELL
/bin/bash
user@pro-Mac:/tmp> csh
user@pro-Mac:/tmp % echo $SHELL
/bin/csh
user@pro-Mac:/tmp % setenv ONLY_IN_CSH true  <—-  set an environment variable from the child csh

user@pro-Mac:/tmp % echo $ONLY_IN_CSH
true
user@pro-Mac:/tmp % suspend   <—- instead of exiting csh “suspend” it
                                      (and it becomes a background job)

[1]+  Stopped                 csh
user@pro-Mac:/tmp> jobs   <— - the ‘jobs” command shows csh is job 1
[1]+  Stopped                 csh

user@pro-Mac:/tmp> !echo:p <— - the last echo command in the bash shell history is the one above to
                                    display $SHELL and not the last echo
                                    command in csh to display $ONLY_IN_CSH
echo $SHELL

user@pro-Mac:/tmp> echo $ONLY_IN_CSH  <—- environment variable in child csh unknown in
                                                   parent bash

user@pro-Mac:/tmp> fg %1  <—-  return to csh (bring csh job to the foreground with fg cmd
csh
user@pro-Mac:/tmp % !echo:p  <— - last echo command in csh history
echo $ONLY_IN_CSH
user@pro-Mac:/tmp % echo $ONLY_IN_CSH
true
user@pro-Mac:/tmp % suspend

[1]+  Stopped                 csh
user@pro-Mac:/tmp> export ONLY_IN_BASH=true  <— - set new environment variable in parent
                                                          bash shell
user@pro-Mac:/tmp> echo $ONLY_IN_BASH
true
user@pro-Mac:/tmp> fg %1. <— - return to child csh
csh
user@pro-Mac:/tmp % echo $ONLY_IN_BASH. <—- new environment variable in parent shell
                                                     unknown to existing child shell
ONLY_IN_BASH: Undefined variable.
```

SHELL - multiple shell environments - suspend and return

Instead of exiting the csh started from bash in the example above you can **suspend a shell** and return to it (with the fg = foreground command)

In this example, the **bash shell** is the **parent process** or parent shell and the **csh** is the **child process** or child shell

Variables exported into the parent shell will be inherited by the child shell UPON its launch

Variables exported into the child shell will not be seen by the parent shell

Variables exported into the parent shell will NOT be inherited by the child shell AFTER the child shell is launched

You cannot suspend a login shell (the original shell launched for the terminal window where SHLVL=1)

SHELL - compare sh and csh code fragments

sh - loop on files matching a wildcard expression

```
#!/bin/sh   <—- name the shell or interpreter to run

files_wildcard="*.dat"

for file in ${files_wildcard}
do
   echo "Found file $file"
done
```

csh - loop on files matching a wildcard expression

```
#!/bin/csh

set files_wildcard=(*.dat)   <—- preface variable assignments with set

foreach file (${files_wildcard})  <—- “foreach - end” replaces “for, do-done”
   echo "Found file $file"
end
```

SHELL - run same script in sh and csh

```
$ touch one.dat two.dat three.dat
```

Run sh script with -x

```
$ sh -x f1.sh
+ files_wildcard='*.dat'
+ for file in '${files_wildcard}'
+ echo 'Found file one.dat'
Found file one.dat
+ for file in '${files_wildcard}'
+ echo 'Found file three.dat'
Found file three.dat
+ for file in '${files_wildcard}'
+ echo 'Found file two.dat'
Found file two.dat
```

Runs csh script with -x

```
$ csh -x f1.csh
set files_wildcard= ( *.dat )
foreach file ( one.dat three.dat two.dat )
echo Found file one.dat
Found file one.dat
end
echo Found file three.dat
Found file three.dat
end
```

SHELL - sh version of script

sh - create a file with a timestamp in the past to compare with newer files

```
#!/bin/sh   <—- name the shell or interpreter to run

# wildcard epxression for files to stat modtime on
files_wildcard="*.dat"
# time in seconds after which to check if files (modtime) changed
wait_time=60
# where to cd to
set subdir="./temp"

# reference file for modification time
file_time_ref=modtime_ref_$$. <— - double dollar sign creates a random number
                                   so we create a uniquely named file for each run

rm -f ${file_time_ref}.   <—- force removal of file incase it already exists
now=`date`                <—- backticks captures the command output into variable

echo "===== From time $now wating $wait_time seconds to stat $files_wildcard files in $subdir
====="
touch ${file_time_ref}  <—- touch creates a new empty file

# set the modification time on the reference file to 1 second ago

if [ ${OSTYPE} == "darwin17" ]; then          <—- If statement testing the builtin
  touch -m -A-000010 ${file_time_ref}             OSTYPE environment variable
elif [ ${OSTYPE} == "linux" ]; then               to check what platform we are on.
  touch -d"-1sec" ${file_time_ref}                Command arguments are different
else                                              on the Mac compared to linux.
  echo "Error: *** platform unknown"
  exit 1       <—-  exit and force a non-zero exit status to indicate an error
fi

ls -l ${file_time_ref}
```

SHELL - csh version of script

```
#!/bin/csh. <— - change name of interpreter to be csh

# wildcard epxression for files to stat modtime on
set files_wildcard="*.dat"  <—- use "set" to preface variable assignments
# time in seconds after which to check if files (modtime) changed
set wait_time=60
# where to cd to
set subdir="./temp"

# reference file for modification time
set file_time_ref=modtime_ref_$$

rm -f ${file_time_ref}
set now=`date`
echo "===== From time $now wating $wait_time seconds to stat $files_wildcard files in $subdir
=====“
touch ${file_time_ref}

# set the modification time on the reference file to 1 second ago
if (${OSTYPE} == "darwin") then              <—-  syntax/grammar of if statement differs
  touch -m -A-000010 ${file_time_ref}
else if (${OSTYPE} == "linux") then
  touch -d"-1sec" ${file_time_ref}
else
  echo "Error: *** platform unknown"
  exit 1
endif

ls -l ${file_time_ref}
```

SHELL - run same script in sh and csh

Run sh script with -x

```
$ sh -x f2.sh
+ files_wildcard='*.dat'
+ wait_time=60
+ subdir=./temp
+ file_time_ref=modtime_ref_10355
+ rm -f modtime_ref_10355
++ date
+ now='Thu Dec  5 00:58:01 EST 2019'
+ echo '===== From time Thu Dec  5 00:58:01 EST 2019 wating 60 seconds to stat *.dat files in ./temp
====='
===== From time Thu Dec  5 00:58:01 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
+ touch modtime_ref_10355
+ '[' darwin17 == darwin17 ']'
+ touch -m -A-000010 modtime_ref_10355
+ ls -l modtime_ref_10355
-rw-r--r--  1 user  staff  0 Dec  5 00:57 modtime_ref_10355
```

Run csh script with -x

```
$ csh -x f2.csh
set files_wildcard=*.dat
set wait_time=60
set subdir=./temp
set file_time_ref=modtime_ref_10363
rm -f modtime_ref_10363
set now=`date`
date
echo ===== From time Thu Dec 5 00:59:35 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
===== From time Thu Dec 5 00:59:35 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
touch modtime_ref_10363
if ( darwin == darwin ) then
touch -m -A-000010 modtime_ref_10363
else if ( darwin == linux ) then
ls -FG -l modtime_ref_10363
-rw-r--r--  1 user  staff  0 Dec  5 00:59 modtime_ref_10363
```

See “watchdog” sh and csh scripts below that test to see if (*.dat) files do not change within a set period of time (wait_time) and send an email notification. Program loops waiting for Ctrl-C interrupt.

```
#!/bin/sh

# absolute or relative path of directory where file mod times should be checked
subdir="./temp"
# wildcard epxression for files to stat modtime on
files_wildcard="*.dat"
# time in seconds after which to check if files (modtime) changed
wait_time=60
# who to send mail to if no files change
mail_list="user@nmr.mgh.harvard.edu"
# run in loop, Ctrl-C to exit if manually run from terminal or if run via cron kill the process with script name
cd ${subdir}

while true
do
  # reference file for modification time
  file_time_ref=modtime_ref_$$
  rm -f ${file_time_ref}
  now=`date`
  echo "===== From time $now waiting $wait_time seconds to stat $files_wildcard files in $subdir ====="
  touch ${file_time_ref}
  # set the modification time on the reference file to 1 second ago
  if [ ${OSTYPE} == "darwin" ]; then
    touch -m -A-000010 ${file_time_ref}
  elif [ ${OSTYPE} == "linux" ]; then
    touch -d"-1sec" ${file_time_ref}
  else
    echo "Error: *** platform unknown"
    exit 1
  fi
  # ls -l ${file_time_ref}
  # remove mod time reference file on interrupt (Ctrl C)
  trap "rm -rf ${file_time_ref}" EXIT

  # echo "Waiting $wait_time seconds"
  sleep $wait_time

  cnt_changed=0
  for file in ${files_wildcard}

  do
    if [ ${file} -nt $file_time_ref ]; then
      echo "File $file HAS CHANGED in $wait_time seconds since $now"
      cnt_changed=`expr $cnt_changed + 1`
      continue
    else
      # echo "File $file has not changed in $wait_time seconds since $now"
      continue
    fi
  done

  if [ $cnt_changed == 0 ]; then
   # use mailx to notify no files changed
   # echo "*** From time $now NO  $files_wildcard FILES CHANGED in $subdir"
   mailx -s "From time $now no $files_wildcard files changed in $subdir" ${mail_list} < /dev/null
  else
   # echo "+++ From time $now $cnt_changed $files_wildcard files changed in $subdir"
   continue
  fi
  rm -f ${file_time_ref}
```

```
#!/bin/csh

# absolute or relative path of directory where file mod times should be checked
set subdir="./temp"
# wildcard epxression for files to stat modtime on
set files_wildcard="*.dat"
# time in seconds after which to check if files (modtime) changed
set wait_time=60
# who to send mail to if no files change
set mail_list="user@nmr.mgh.harvard.edu"

# run in loop, Ctrl-C to exit if manually run from terminal or if run via cron kill the process with script name
cd ${subdir}
while (1)

  # reference file for modification time
  set file_time_ref=modtime_ref_$$
  rm -f ${file_time_ref}
  set now=`date`
  echo "===== From time $now wating $wait_time seconds to stat $files_wildcard files in $subdir ====="
  touch ${file_time_ref}
  # set the modification time on the reference file to 1 second ago
  if (${OSTYPE} == "darwin") then
    touch -m -A-000010 ${file_time_ref}
  else if (${OSTYPE} == "linux") then
    touch -d"-1sec" ${file_time_ref}
  else
    echo "Error: *** platform unknown"
    exit 1
  endif
  # ls -l ${file_time_ref}

  # remove mod time reference file on interrupt (Ctrl C)
  # FIX ME onintr rm -f $file_time_ref WRITE FUNCTION TO DO THIS

  # echo "Waiting $wait_time seconds"
  sleep $wait_time

  # csh cannot compare file times via an if conditional as bash does
  @ cnt = 0
  foreach file (${files_wildcard})
    set grep_output=""
    set grep_output=`find . -type f -newer $file_time_ref | sed 's;\.\/;;g' | grep ${file}`
    if (${grep_output} != "") then
      echo "+++ FILE $file HAS CHANGED in $wait_time seconds since $now"
      @ cnt += 1
    else
      # echo "File $file has not changed in $wait_time seconds since $now"
    endif
  end

  if ($cnt == 0) then
   # use mailx to notify no files changed
   # echo "*** From time $now NO  $files_wildcard FILES CHANGED in $subdir"
   mailx -s "From time $now no $files_wildcard files changed in $subdir" ${mail_list} < /dev/null
  else
   echo "+++ From time $now $cnt $files_wildcard files changed in $subdir"
  endif

  rm -f ${file_time_ref}

end
```

SHELL - run sh script

```
+ subdir=./temp
+ files_wildcard='*.dat'
+ wait_time=3
+ mail_list=rd521@nmr.mgh.harvard.edu
+ cd ./temp
+ true
+ file_time_ref=modtime_ref_17818
+ rm -f modtime_ref_17818
++ date
+ now='Thu Dec  5 14:02:25 EST 2019'
+ echo '===== From time Thu Dec  5 14:02:25 EST 2019 wating 3 seconds to stat *.dat files in ./
temp ====='
===== From time Thu Dec  5 14:02:25 EST 2019 wating 3 seconds to stat *.dat files in ./temp
=====
+ touch modtime_ref_17818
+ '[' darwin17 == darwin17 ']'
+ touch -m -A-000010 modtime_ref_17818
+ trap 'rm -rf modtime_ref_17818' EXIT
+ sleep 3
+ cnt_changed=0
+ for file in '${files_wildcard}'
+ '[' '*.dat' -nt modtime_ref_17818 ']'
+ continue
+ '[' 0 == 0 ']'
+ mailx -s 'From time Thu Dec  5 14:02:25 EST 2019 no *.dat files changed in ./temp'
user@nmr.mgh.harvard.edu
Null message body; hope that's ok
+ rm -f modtime_ref_17818
+ true
+ file_time_ref=modtime_ref_17818
+ rm -f modtime_ref_17818
++ date
+ now='Thu Dec  5 14:02:28 EST 2019'
```

SHELL - run csh script (after touching 2 *.dat files)

```
echo ===== From time Mon Dec 9 21:52:39 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
===== From time Mon Dec 9 21:52:39 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
touch modtime_ref_26116
if ( darwin == darwin ) then
touch -m -A-000010 modtime_ref_26116
else if ( darwin == linux ) then
sleep 60
@ cnt = 0
foreach file ( *.dat )
set grep_output=
set grep_output=`find . -type f -newer $file_time_ref | sed 's;\.\/;;g' | grep ${file}`
find . -type f -newer modtime_ref_26116
sed s;\.\/;;g
grep one.dat
if ( one.dat !=  ) then
echo +++ FILE one.dat HAS CHANGED in 60 seconds since Mon Dec 9 21:52:39 EST 2019
+++ FILE one.dat HAS CHANGED in 60 seconds since Mon Dec 9 21:52:39 EST 2019
@ cnt += 1
else
end
set grep_output=
set grep_output=`find . -type f -newer $file_time_ref | sed 's;\.\/;;g' | grep ${file}`
find . -type f -newer modtime_ref_26116
sed s;\.\/;;g
grep three.dat
if ( !=  ) then
endif
end
set grep_output=
set grep_output=`find . -type f -newer $file_time_ref | sed 's;\.\/;;g' | grep ${file}`
find . -type f -newer modtime_ref_26116
sed s;\.\/;;g
grep two.dat
if ( two.dat !=  ) then
echo +++ FILE two.dat HAS CHANGED in 60 seconds since Mon Dec 9 21:52:39 EST 2019
+++ FILE two.dat HAS CHANGED in 60 seconds since Mon Dec 9 21:52:39 EST 2019
@ cnt += 1
else
end
if ( 2 == 0 ) then
echo +++ From time Mon Dec 9 21:52:39 EST 2019 2 *.dat files changed in ./temp
+++ From time Mon Dec 9 21:52:39 EST 2019 2 *.dat files changed in ./temp
endif
rm -f modtime_ref_26116
end
while ( 1 )
set file_time_ref=modtime_ref_26116
rm -f modtime_ref_26116
set now=`date`
date
echo ===== From time Mon Dec 9 21:53:39 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
===== From time Mon Dec 9 21:53:39 EST 2019 wating 60 seconds to stat *.dat files in ./temp =====
touch modtime_ref_26116
if ( darwin == darwin ) then
touch -m -A-000010 modtime_ref_26116
else if ( darwin == linux ) then
```

SHELL - bourne (sh), bash, zsh

BOURNE AND BASH

Most Bourne shell scripts should run under bash. So even if /bin/sh is Bourne shell (and not bash), then it should not matter if your use /bin/sh or /bin/bash to run a Bourne shell script

But since Bourne shell is not as POSIX compliant, then bash shell scripts may not run using Bourne shell, e.g., if Bourne shell is /bin/sh

WHERE /bin/sh IS NOT bash

On Debian and Ubuntu systems /bin/sh is a link to dash shell

Non-linux systems, e.g., BSD (OpenBSD, FreeBSD)

ZSH ADDITIONS

File globbing

Spelling correction

Directory aliases (much like ~ or ..)

Loadable modules, like socket controls or an FTP client

You can use zsh as a replacement for Bash (put zsh in compatibility mode)

Startup/shutdown scripts via zshenv, zprofile, zshrc, zlogin, and zlogout

git command completion

Added command line expansion - enter cd /u/lo/b, press tab, and it will be completed to cd /usr/local/bin if it is the only matching pattern

SHELL - chaining commands with shell operators ; && ||

A single semicolon, double ampersand or double vertical bar can be used to chain together commands

Note the order of these operators will affect the status returned by the builtin status variable ($? in bash and $status in csh)

```
cmd_a  ;  cmd_b
cmd_a  &&  cmd_b  (and operator)
cmd_a && cmd_b … ||  cmd_c
```

```
user@pro-Mac:/tmp> true ; date.  <—- both cmds succeed, but status returned only for last cmd
Sun Dec  8 20:51:19 EST 2019
user@pro-Mac:/tmp> echo $?
0

user@pro-Mac:/tmp> false ; date <—- 1st cmd fails, 2nd cmd succeeds returning status=0 success
Sun Dec  8 20:51:26 EST 2019
user@pro-Mac:/tmp> echo $?
0

user@pro-Mac:/tmp> true && date <— both cmds succeed
Sun Dec  8 20:51:36 EST 2019
user@pro-Mac:/tmp> echo $?
0

user@pro-Mac:/tmp> false && date <—- 1st cmd returns status=1 failure so 2nd cmd never runs
user@pro-Mac:/tmp> echo $?
1

user@pro-Mac:/tmp> true && date || whoami <—- both cmds succeed
Sun Dec  8 20:54:59 EST 2019
user@pro-Mac:/tmp> echo $?
0

user@pro-Mac:/tmp> false && date || whoami <—-1st cmd returns status=1 so 2nd cmd never runs,
                                              but because non-zero status prior to || the 3rd
                                               cmd runs and succeeds retuning status=0
user
user@pro-Mac:/tmp> echo $?
0
```

SHELL - chaining commands with shell operators ; && ||

cmd_a ; cmd_b
Run cmd_a and then cmd_b, regardless of the success or failure of cmd_a

cmd_a && cmd_b (and operator)
Run cmd_b only if cmd_a succeeded

cmd_a && cmd_b … || cmd_c
Run cmd_c only if cmd_a or cmd_b failed

Beware that not using && to chain commands together can result in disaster with commands like rm -rf *

```
user@pro-Mac:/tmp> mkdir help
user@pro-Mac:/tmp> cd help
user@pro-Mac:/tmp/help> touch one two three
user@pro-Mac:/tmp/help> ls
one  three  two

user@pro-Mac:/tmp/help> cd /tmp2; rm -f *
```

<—-Since /tmp2 does not exist, the shell cd's to the current working directory. Then because the next command will run whether or not the first command succeeded or failed, all files are removed in the current working directory!

```
bash: cd: /tmp2: No such file or directory
user@pro-Mac:/tmp/help> ls

user@pro-Mac:/tmp/help> touch one two three

user@pro-Mac:/tmp/help> cd /tmp2 && rm -f *
```

<—- Use of && instead of ; prevents files from accidental deletion

```
bash: cd: /tmp2: No such file or directory
user@pro-Mac:/tmp/help> ls
one    three  two
```

SHELL - piping commands together

A **single vertical bar known as a “pipe”** sends the output of the first command to be the input of the second command

cmd_a | cmd_b

List the shell environment variables in sorted (lexigraphical) order.

Take the output from the env command and pipe it to the input of the grep command

```
$ env | sort
HOME=/Users/user
HOST=pro-Mac.local
HOSTTYPE=unknown
LANG=en_US.UTF-8
LOGNAME=user
MACHTYPE=x86_64
OLDPWD=/Volumes/partition_2/freesurfer_src
OSTYPE=darwin
PATH=/usr/bin:/bin:/usr/sbin:/sbin:/opt/X11/bin:/usr/local/bin:/Users/user/bin
PWD=/tmp
SECURITYSESSIONID=186a7
SHELL=/bin/csh
SHLVL=2
```

List the order of entries in PATH with 1 entry per line (useful if you have a long convoluted PATH to review)

Take the output from env command and pipe it to grep command to find what PATH is set to.
Take the output from the grep command and pipe it to the sed command where =: will be substituted for :
(Add the colon as part of the substitution because in the next command we will substitute on the colon)
Take the output from the sed command and pipe it to the input of the translate (tr) command where any instance of : will be replaced with a carriage return line feed \n

```
$ env | grep PATH | sed 's/=/=:/' | tr -s ':' '\n'
PATH=
/usr/bin
/bin
/usr/sbin
/sbin
/opt/X11/bin
/usr/local/bin
/Users/user/bin
```

… without the sed command …

```
PATH=/usr/bin
/bin
/usr/sbin
/sbin
/opt/X11/bin
/usr/local/bin
/Users/user/bin
```

SHELL - piping commands together

Use the **xargs command** in a pipeline to run a command that does not normally accept piped input

cmd_a | xargs cmd_b *<run cmd_b on each line output from cmd_a>* | cmd_c

Say you wanted to find every binary file under freesurfer/bin (which contains executables that are both shell scripts and binaries). On linux the "file" command will identify binaries as "ELF" type and on the Mac they will be identified as "Mach-O" type. One way to do this is to find all files (that are not subdirectories), run the file command on *each* file output from the find command via the xargs command and then grep the output from the file command to find only ELF or Mach-o type files.

Here is the single line output from the file command (on linux) on a freesurfer binary,

```
$ cd  /usr/local/freesurfer/dev/bin
$ file mri_convert
mri_convert: ELF 64-bit LSB executable, x86-64, version 1 (GNU/Linux), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, stripped
```
… and on the Mac …
```
mri_convert: Mach-O 64-bit executable x86_64
```

If you only want the binary file name, then strip out everything after the colon with the sed command. Or the sed command at the end of the pipeline that for the first colon followed by any characters (the wildcard .*) - substitute nothing. Or remove all text starting with the first colon and anything that follows it. The "/" char is the delimiter to separate patterns. The sort command is added last to sort the list.

```
$ cd /usr/local/freesurfer/dev/bin

# find all linux ELF binaries
$ find . -type f | xargs file | grep ELF | sed 's/:.*//' | sort

# find all darwin Mach-O binaries
$ find . -type f | xargs file | grep "Mach-O" | sed 's/:.*//' | sort
```

Since the output of the file command outputs only 1 line per binary found then (after the grep command) if you only want to count the binaries, the last command in the pipeline can be the word count command with the -l option to count the total number of lines.

```
# linux - count ELF binaries
$ find . -type f | xargs file | grep ELF | wc -l

# Mac - count Mach-O binaries
$ find . -type f | xargs file | grep "Mach-O" | wc -l
```

SHELL - piping commands together

Use the **awk command** with command pipes to dynamically generate scripts with shell command line arguments for subsequent execution

It's often useful to find a subset of files (with the find command) and then subsequently process the list of files e.g., copy them to a different destination. The awk text processing tool (named after the initials of its authors) is a scripting language in its own right, but can provide "1 liners" to generate scripts.

For example, say you had a sandbox with build, makefiles, etc., you had extensively changed. Maybe you do not want to commit or push he changes until you had tried them in different sandboxes. To isolate the files, and copy then into another sandbox tree, you could do the following.

```
$ cd <my sandbox base>/freesurfer
```

```
$ find . -name "CMakeLists.txt"
./CMakeLists.txt
./diffusion_tool/CMakeLists.txt
./distribution/average/Buckner_JNeurophysiol11_MNI152/CMakeLists.txt
… etc …
```

The find command generates a line of output for each file. To copy all the cmake files we want to take each file output by the find command and create a script that looks like,

```
$ cp -p -f <path 1>/<file>  <path 2>/<file>
```

The file name will contain the *relative* path under ./freesurfer, so if we are copying files between sandbox trees, then it's likely useful to have <path 1> and <path 2> as arguments to a shell script to copy files. A script called "copy.sh" might look like,

```
#!/bin/bash
cp  -p  -f  $1/CMakeLists.txt  $2/CMakeLists.txt
cp  -p  -f  $1/diffusion_tool/CMakeLists.txt  $2/diffusion_tool/CMakeLists.txt
cp  -p  -f  $1/distribution/average/Buckner_JNeurophysiol11_MNI152/CMakeLists.txt  $2/distribution/average/
Buckner_JNeurophysiol11_MNI152/CMakeLists.txt
… etc …
```

The variable $1 is the first command line argument (path to the existing sandbox) and $2 is the second command line argument (path to the destination sandbox where files should be copied to). The copy.sh script would be run with those command line arguments as,

```
$ sh  -x  copy.sh  /Volumes/sandbox_1/freesurfer  /Volumes/sandbox_2/freesurfer
… and the argument variables $1 and $2 on first line in the script would be expanded to run as …
cp  -p  -f  /Volumes/sandbox_1/freesurfer/CMakeLists.txt  /Volumes/sandbox_2/freesurfer/CMakeLists.txt
```

The awk language can generate the above script by processing arguments with dollar sign variables. By default awk assumes text is separated by whitespace (or the IFS or "inter-field separator" in awk is set to whitespace by default). In awk the $0 variable is the *entire*line* of text input and $1 is the first text field from the beginning of the line up to the IFS; $2 is the second text field between the first 2 instances of the IFS, etc… to $3, $4 for however many fields of text exist on a line separated by instances of the IFS.

SHELL - piping commands together

To create the copy.sh script, pipe the output of the find command into an awk command that will embed the shell command line arguments along with awk processing it's own command line arguments.

$ **find . -name "CMakeLists.txt" | awk '{print "cp -p -f $1/"$0" $2/"$0}'**

The awk commands are contained between curly braces {} and are protected from shell parsing by being enclosed within single quotes.

awk '{<commands>}'

Within the curly braces anything in double quotes will be literally printed by awk so the following would print "some text" in awk,

awk '{print "some text"}'

The following would print "some text/$1"

awk '{print "some text/$1"}'

However, within the curly braces, any dollar sign variables *not*in*double*quotes* will be expanded by awk using what is piped in from the find command. Since $0 is not in double quotes, then it expands to the entire line piped in by find.

The complete awk command **'{print "cp -p -f $1/"$0" $2/"$0}'** is read as,
print "cp -p -f $1/"
expand $0 to be the output of the complete text line from find
print " $2/"
expand $0 to be the output of the complete text line from find

For the first line output from the find command, ./CMakeLists.txt, the output from find piped to the awk,

$ **find . -name "CMakeLists.txt" | awk '{print "cp -p -f $1/"$0" $2/"$0}'**
… produces the first line of output…
cp -p -f $1/./CMakeLists.txt $2/./CMakeLists.txt

While this command will work, it might be nice to remove the leading dot slash from the output of find. This can be done by inserting a sed command between the output of find and the input to awk to substitute for the "beginning of line dot slash" or "^\.\/" - no characters at all. (The chars ./ should be escaped with backslash to yield \.\/ and ^ is beginning of line).

$ **find . -name "CMakeLists.txt" | sed 's;^\.\/;;' | awk '{print "cp -p -f $1/"$0" $2/"$0}'**
… producing the first line of output …
cp -p -f $1/CMakeLists.txt $2/CMakeLists.txt

To create the script copy.sh we only need to redirect the output from stdout to a file by that name using the ">"

$ **find . -name "CMakeLists.txt" | sed 's;^\.\/;;' | awk '{print "cp -p -f $1/"$0" $2/"$0}' > copy.sh**
… and the script would be run using the $1 and $2 arguments to the shell as,
$ **sh -x copy.sh <path to sandbox 1> <path to sandbox 2>**

SHELL - conclusions

POSIX compliant bash and shell variants (dash, ash, etc…) are most common now compared to older non-POSIX shells like csh or Bourne (as /bin/sh) to the point where /bin/sh is actually bash (via a soft link or just another copy of /bin/bash)

Shell development and variations have continued as recently as 1990 with offerings like zsh

Shells have extensive features that let you program and manipulate the shell environment starting with the login shell (shell init files) to set and export common environment variables like PATH, OSTYPE, USER, etc.

Shells provide builtin functions and the ability to nest, suspend, and resume running child or sub-shells where each shell has its own unique environment (variables) and saved command history

Shells provide multiple ways to exec commands including alias shortcuts and chaining together commands via a single semicolon, two ampersands, two vertical bars and a single vertical bar for the pipe operator

Some shell environment must be used when languages like C, Java, python, etc. exec a process. The default shell is often /bin/sh or /bin/bash but it is good practice to explicitly set /bin/bash if the exec call allows it

System administration scripts are usually written in sh/bash because if your machine can only boot into single user mode (no GUI desktop), then the only shell you may have is sh/bash, e.g., to unmount, repair and mount the boot partition. You will probably not be able to avoid having to know something about bash scripting if you work at a systems level

The C-shell is considered by “experts” to be fine for use as an interactive shell, but frowned upon for shell scripting - bash is preferred by many engineers for shell scripting (csh is not pre-installed on all linux systems).

You can find shell programs that are thousands of lines long, but with the advent of scripting languages such as Perl and Python, many organizations switched to writing scripts in these languages barring some system dependency upon using sh/bash scripting.