INTRO TO LINUX

A Crash Course on Using the Best OS Ever Invented

Trevor Vannoy

EELE 466 | Jan. 10, 2018

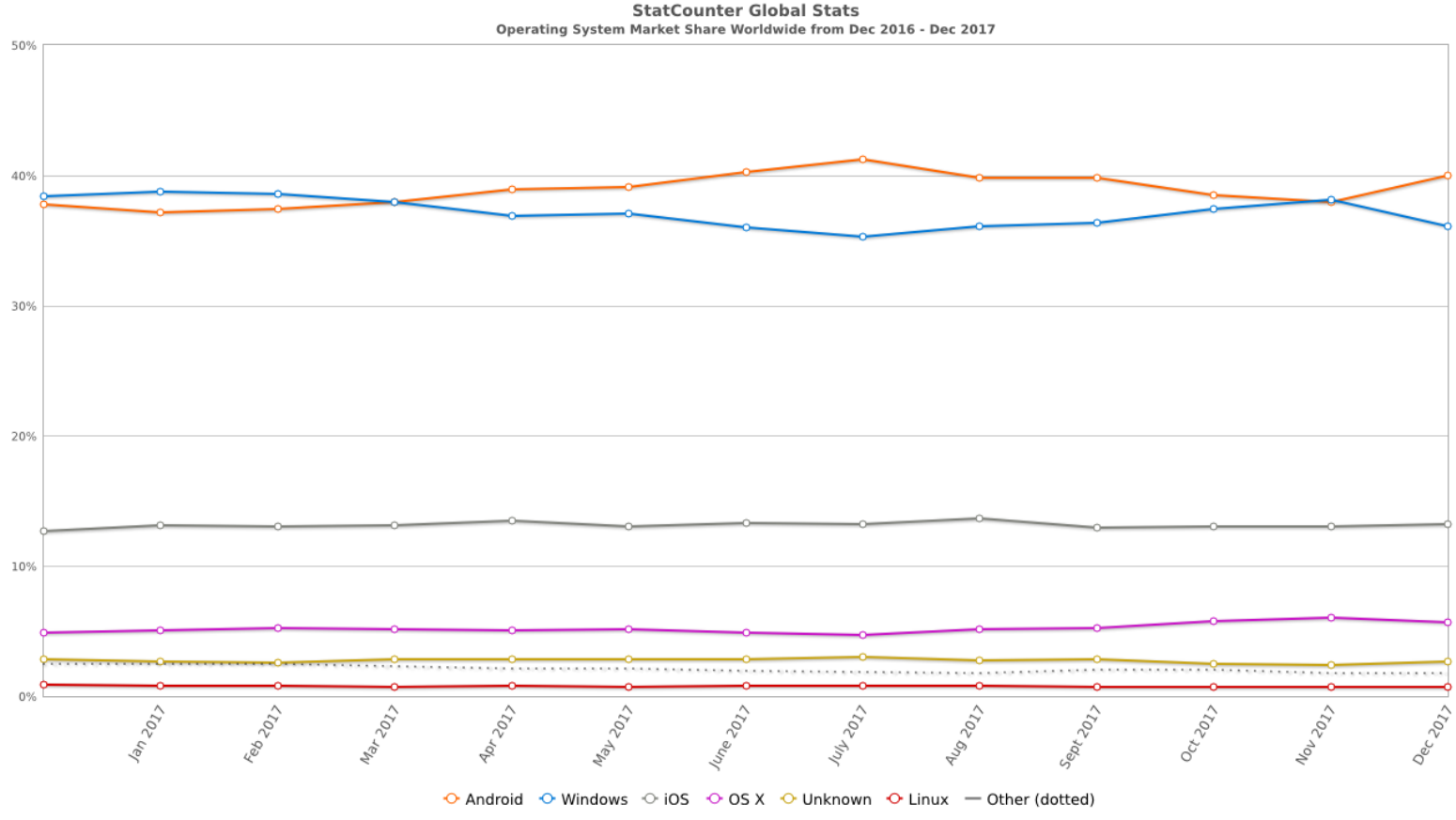
OUTLINE

- What is Linux and who uses it?
- History
- Unix philosophy
- File system overview
- Using the terminal
- Shell scripts
- Embedded development
- References

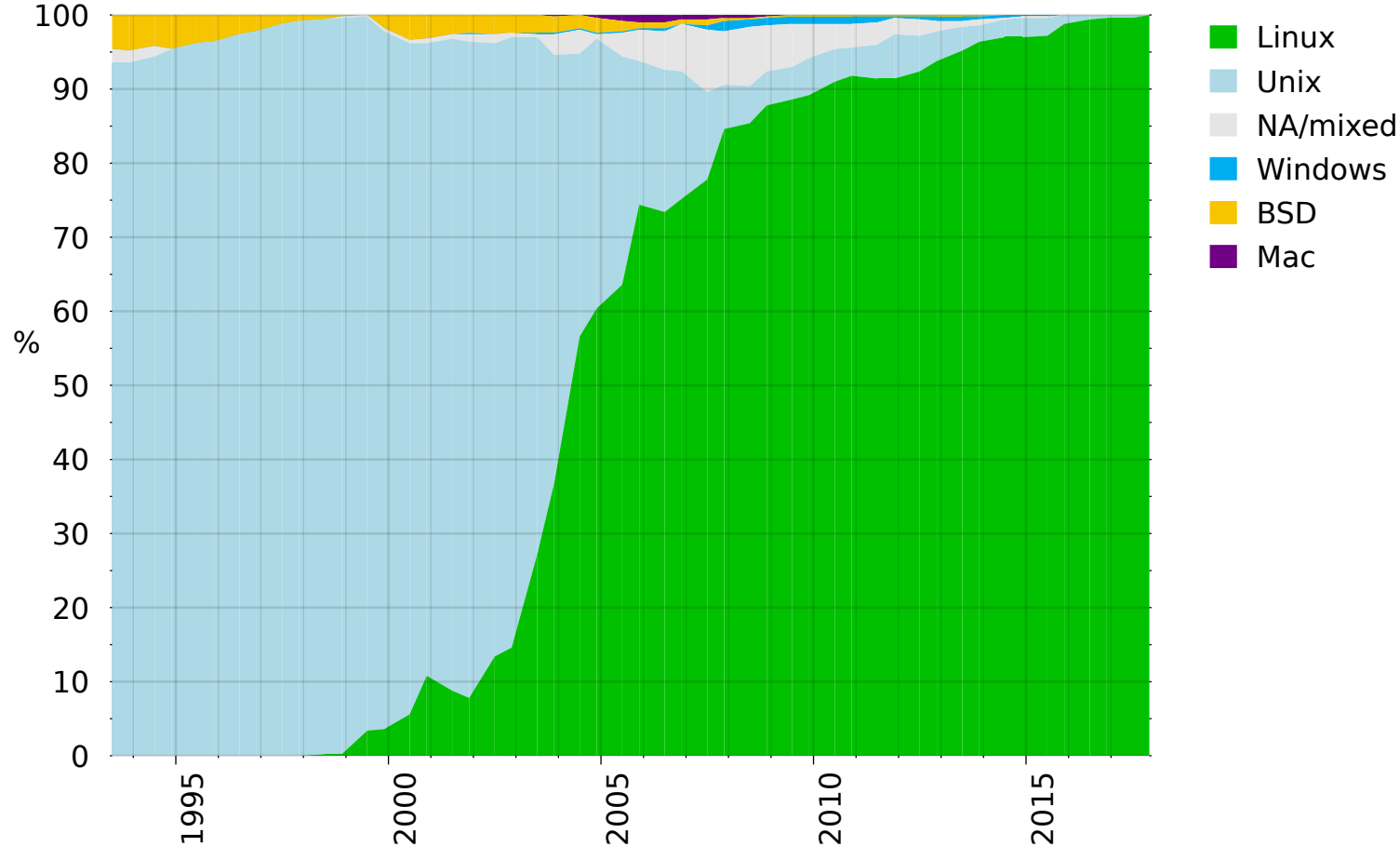
WHAT IS LINUX AND WHO USES IT?


Linux Mint
from freedom came elegance
fedora
archlinux
redhat

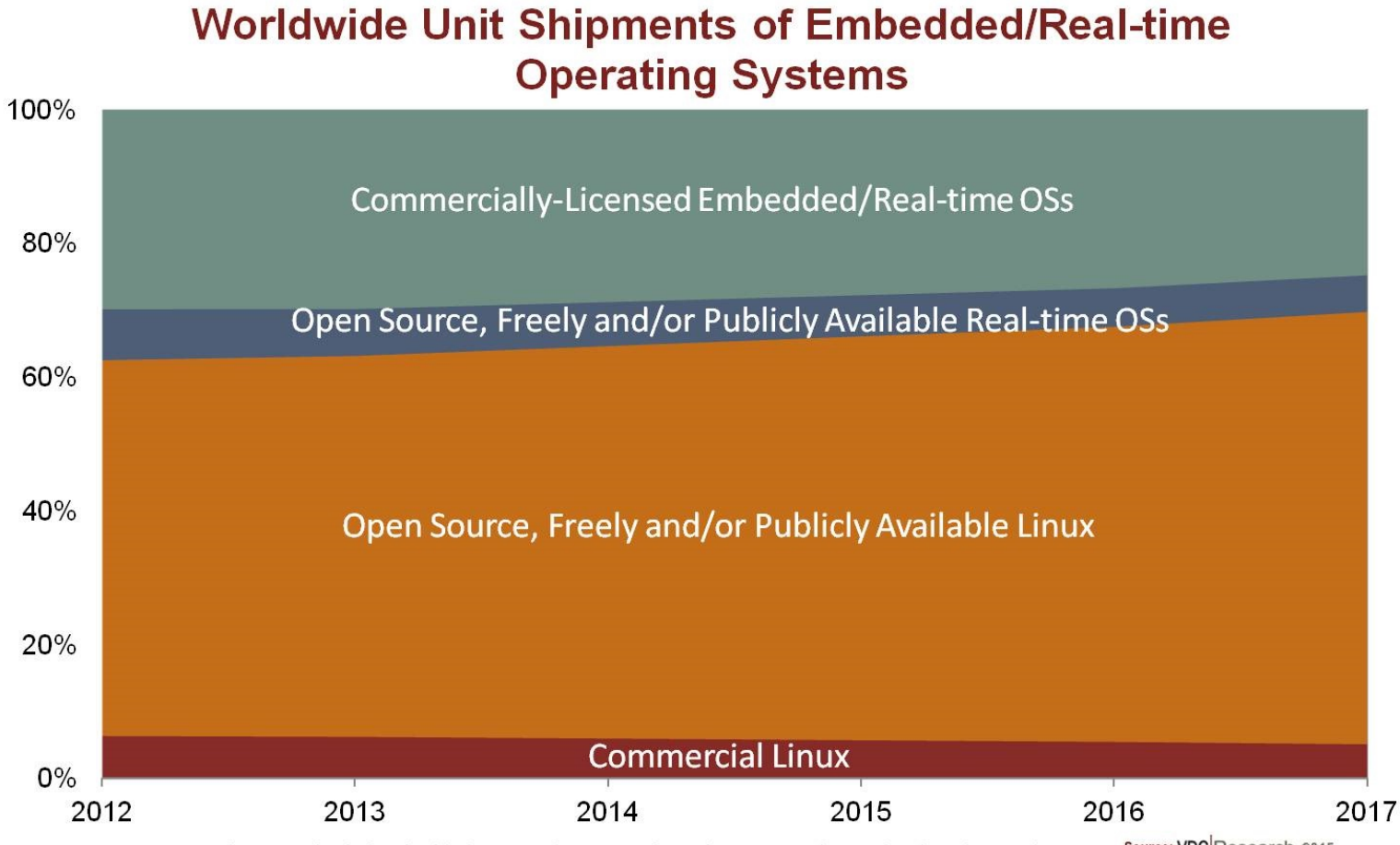
OS MARKET SHARE



TOP 500 SUPERCOMPUTERS



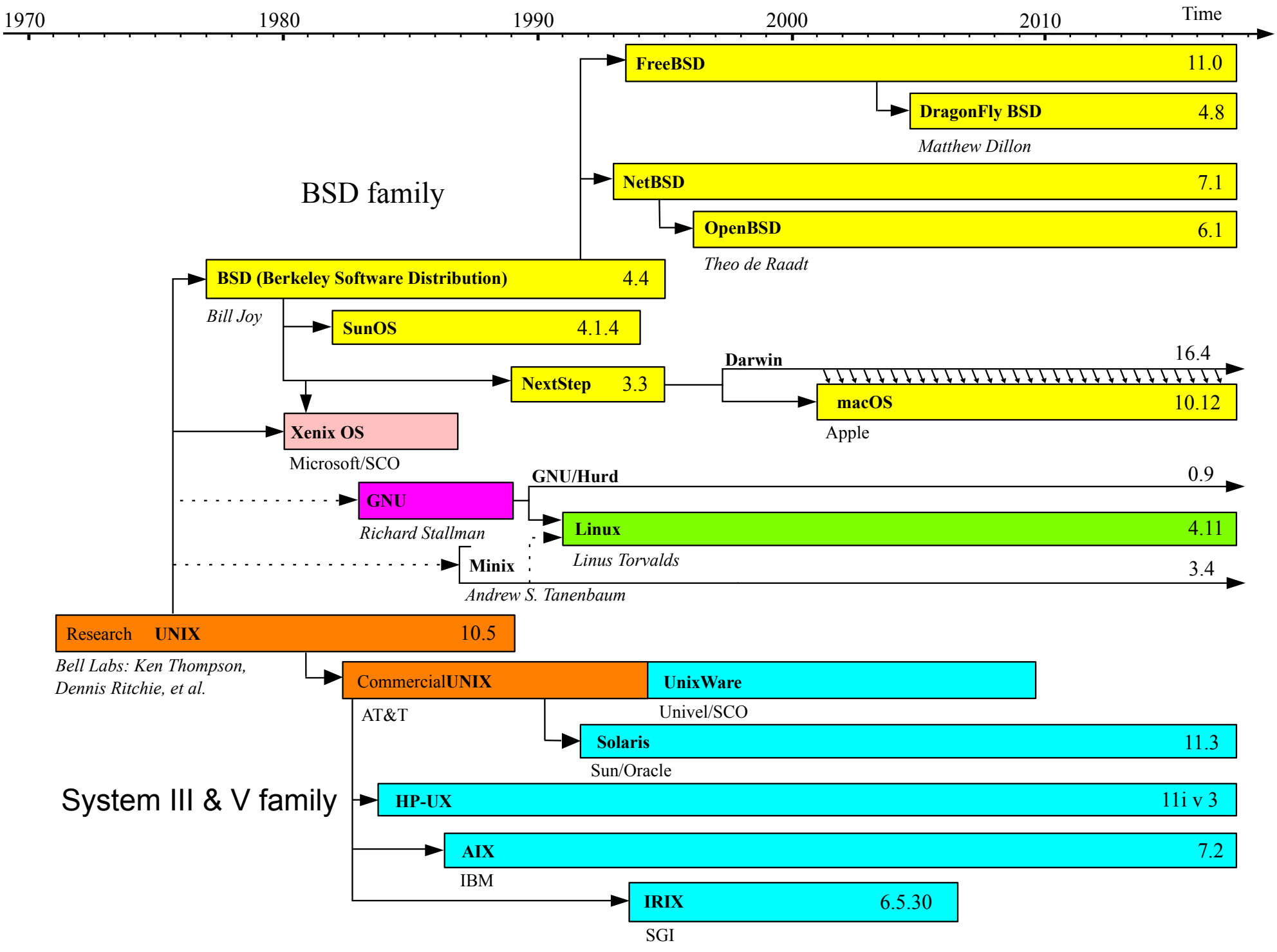
EMBEDDED SPACE



Note: More than one-third of embedded projects feature no formal OS or an in-house developed OS and are not depicted in the chart above.

Source: VDC Research, 2015

HISTORY


1970
1980
1990
2000
2010
Time
BSD family
FreeBSD 11.0
DragonFly BSD 4.8
Matthew Dillon
NetBSD 7.1
OpenBSD 6.1
Theo de Raadt
BSD (Berkeley Software Distribution) 4.4
Bill Joy
SunOS 4.1.4
Darwin
16.4
NextStep 3.3
macOS 10.12
Apple
Xenix OS
Microsoft/SCO
GNU/Hurd
0.9
GNU
Richard Stallman
Linux 4.11
Linus Torvalds
Minix
3.4
Andrew S. Tanenbaum
Research UNIX 10.5
Bell Labs: Ken Thompson,
Dennis Ritchie, et al.
CommercialUNIX
UnixWare
AT&T
Univel/SCO
Solaris 11.3
Sun/Oracle
System III & V family
HP-UX 11i v 3
AIX 7.2
IBM
IRIX 6.5.30
SGI

UNIX PHILOSOPHY

SIMPLICITY, MINIMALISM, AND COMPOSABILITY

- Write programs that do one thing and do it well.
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.
- Small is beautiful.
- Build a prototype as soon as possible.
- Choose portability over efficiency.
- Store data in flat text files.

EVERYTHING IS A FILE (DESCRIPTOR)

- most resources are exposed as a stream of bytes
- this allows common tools to operate on different things
- most configuration is contained in plain text files
- hardware and system properties can be exposed in the filesystem

FILE SYSTEM OVERVIEW

PERMISSIONS

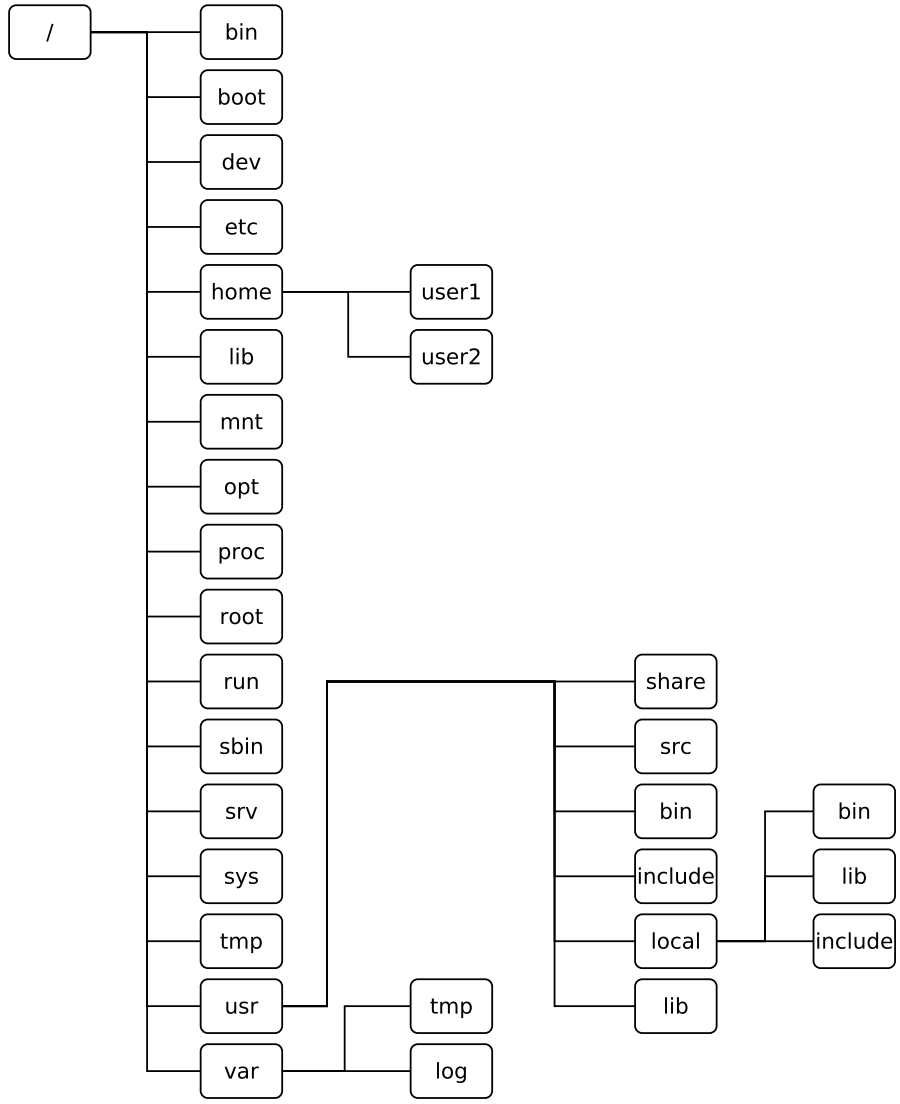
classes:

- user
- group
- other

modes:

- read
- write
- execute

FILE SYSTEM HIERARCHY



directory	purpose
/	file system root
/boot	boot loader files
/etc	system-specifc config files
/home	user home directories (personal files, settings)
/root	root user home directory
/tmp	small temporary files (typically flushed at boot)
/run	tmpfs file system for system packages to place runtime data in

directory	purpose
/usr	read-only user data
/usr/bin	binaries and executables
/usr/lib	libraries for binaries
/usr/include	C/C++ header files
/usr/share	shared resources (documentation, fonts, themes, etc)
/usr/local	third-party packages (user-installed programs, etc.)

directory	purpose
/bin	essential binaries (symlinked to /usr/bin in systems using systemd)
/sbin	essential system binaries (symlinked to /usr/bin in systemd)
/usr/sbin	non-essential system binaries (symlinked to /usr/bin in systemd)
/lib	libraries for essential binarires (symlinked to /usr/lib in systemd)

directory	purpose
/var	persistant, variable system data
/var/log	persistant system logs
/var/tmp	large temporary files
/dev	root directory for device nodes
/proc	virtual kernel file system exposing the process list
/proc/sys	exposes kernel tunables
/sys	virutal kernel file system exposing discovered devices/drivers

directory	purpose
~	alias for /home/*current_user*
~/.config	user application configuration
/opt	optional packages (same purpose as /usr/local)
/media	temporary mount directory for removable devices
/mnt	temporary mount directory for file systems

USING THE TERMINAL

NAVIGATING DIRECTORIES

- cd : change directory
- ls : list files and directories
- pwd : print current directory

EXAMPLES

move to a subdirectory of the current dirctory

```
cd dir1
```

move up a directory

```
cd ../
```

use long listing format

```
ls -l
```

include hidden entries

```
ls -a
```

WORKING WITH FILES AND DIRECTORIES

- **mkdir** : make a directory
- **rm** : remove a files and directories
- **mv** : move files and directories
- **cp** : copy files
- **touch** : change file timestamps (can be used to create a blank file)
- **find** : find files and directories

EXAMPLES

remove directories and files recursively, including empty directories

```
rm -rf
```

copy a directory recursively

```
cp -R
```

move files using wildcards

```
mv *.c src/
```

find files containing a string, starting from the current directory

```
find . -iname '*foo*' -type f
```

DATA MANIPULATION

- **echo** : echo a string to stdout
- **cat** : concatenate files and print on stdout
- **less** : paginate files
- **tail** : get end of file
- **head** : get beginning of file
- **grep** : print lines matching a pattern (**g**lobal **r**egular **e**xpression **p**rint)

EXAMPLES

recursively search for a string in a directory

```
grep -r foobar .
```

print line numbers where a string was found

```
grep -n foobar file
```

search using Perl regular expressions

```
grep -P '\w[a-z]l{1,2}' *
```

CHANGING OWNERSHIP AND PERMISSIONS

- chown : change owner
- chmod : change mode bits

EXAMPLES

change user and group of a file

```
chwon new_user:new_group file
```

add exectute permissions to everybody

```
chmod +x file
```

set permissions to (rwx, rx, rx) for (user, group, other)

```
chmod 755 file
```

set permissions to (rw, r, r) for (user, group, other)

```
chmod 644 file
```

WORKING WITH PROCESSES

- ps : snapshot of the current processes
- top or htop : process viewer
- kill : terminate a process (typically by process ID)
- killall : terminate a process by name

EXAMPLES

view all processes from all users

```
ps aux
```

VIEWING LOGS

- dmesg : display the kernel message buffer
- journalctl : display systemd journals
- cat log files in /var/log

EXAMPLES

show errors from current boot

```
journalctl -p err -b
```

I/O REDIRECTION

- `>` : redirect stdout to a file
- `>>` : append stdout to a file
- `2>` and `2>>` : redirect stderr
- `&>` and `&>>` : redirect stdout and stderr
- `<` : accept input from a file
- `` `command` `` : command substituion. use output of one command in another
- `<< EOF` : read from a string literal, using a here document, until EOF

EXAMPLES

create a signle-line file

```
echo 'some text' > some_file
```

create a multi-line file

```
cat >> file1 << EOF
Culpa est in repellat inventore veniam.
Id totam dolorem consectetur voluptates
EOF
```

append program output to a file

```
./a.out >> output_file
```

remove files whose filenames are in a file

```
rm `cat filenames`
```

PIPES

Pipes connect the stdout of one process to the stdin of another.

| is the pipe symbol.

```
dmesg | grep ACPI
```

```
dmesg | tail
```

```
cat words | tr " " "\n" | sort
```

```
ps aux | grep process_name
```

ALIASES

Replaces a word by another string. They can be used to abbreviate commands.

```
alias ll="ls -l"
alias dog=cat
```

They need to be put in a config file (e.g. ~/.bashrc, ~/.zshrc) to be persistant.

ENVIRONMENT VARIABLES

Just like other languages, shell has variables.

Environment variables affect the current environment and how processes are run.

```
export PATH=$PATH:/opt/bin
```

```
RUST_BACKTRACE=1 ./test
```

PROCESS CONTROL

- command **&** : run command in the background
- command1 **&&** command2 : run command1, then run command2 if command1 exited without error

Ex:

```
./long_script.sh &
```

GETTING HELP

A.K.A RTFM

- -h or --help
  - some tools only have one or the other
- man pages

Ex:

```
cp --help
```

```
man mv
```

SHELL SCRIPTS

AUTOMATION

We can put a bunch of shell commands into a script!

- The script must start with "#!/bin/sh" or "#!/bin/bash" "#!/bin/zsh" etc...
- The script needs to be executable:

```
chmod +x script.sh
```

- Execute in the normal way:

```
./script.sh
```

EXAMPLES

hello, world

```
#!/bin/sh
echo "hello, world!'
```

we can use variables

```
#!/bin/sh

NUM_CATS=`grep -o cat words | wc -w`
NUM_DOGS=`grep -o dog words | wc -w`
NUM_CATDOGS=`grep -o catdog words | wc -w`

echo "I found $NUM_CATS cats and $NUM_DOGS dogs."
echo "That equals $((NUM_CATS + NUM_DOGS)) cats and dogs."
echo "I found $NUM_CATDOGS catdogs!"
```

IF STATEMENTS

if statements are pretty similar to any other language. Unforunately, tests return 0 for true and 1 for false.

```
#!/bin/bash

if [[ test condition ]] ; then
    echo "if"
elif [[ test condition ]] ; then
    echo "elif"
else
    echo "else"
fi
```

FOR LOOPS

general syntax

```
for expr; do
    echo "do something"
done
```

using glob patterns

```
for file in *.txt ; do
  mv "$file" "$file.bak"
done
```

using ranges

```
for i in {0..10}
do
    echo $i
done
```

WHILE LOOPS

```
#!/bin/bash
X=0
while [ $X -le 10 ]
do
    echo $X
    X=$((X+1))
done
```

EMBEDDED DEVELOPMENT

WRITING SD CARD IMAGES

You can use dd to write sd card images, among other things.

```
dd if=image of=/dev/sdx bs=4M status=progress && sync
```

Sync is typically used to make sure any cached writes are synchronized to the storage.

CONTROLLING HARDWARE THROUGH THE /SYS INTERFACE

Because most hardware devices can be exposed through the /sys virtual file system, we can probe and control our hardware with simple shell commands!

This isn't always the best way to accomplish a task, but it's simple.

Ex:

```
echo 0 > /sys/class/leds/hps_led0/brightness
echo 1 > /sys/class/leds/hps_led0/brightness
```

REFERENCES

SHELL SCRIPTING

https://arachnoid.com/linux/shell_programming.html
http://www.panix.com/~elflord/unix/bash-tute.html
http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
https://linuxconfig.org/bash-scripting-tutorial-for-beginners
https://en.wikibooks.org/wiki/Bash_Shell_Scripting#Shell_arithmetic
http://matt.might.net/articles/bash-by-example/
http://tldp.org/LDP/Bash-Beginners-Guide/html/

SYSFS AND PROCFS

https://www.kernel.org/doc/Documentation/gpio/sysfs.txt
https://en.wikipedia.org/wiki/Sysfs
http://kroah.com/log/blog/2013/06/26/how-to-create-a-sysfs-file-correctly/ https://en.wikipedia.org/wiki/Procfs
https://kernelnewbies.org/Documents/Kernel-Docbooks?action=AttachFile&do=get&target=procfs-guide_2.6.29.pdf

FILE SYSTEM

https://en.wikipedia.org/wiki/File_system_permissions#Traditional_Uni
http://jlk.fjfi.cvut.cz/arch/manpages/man/file-hierarchy.7
https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard