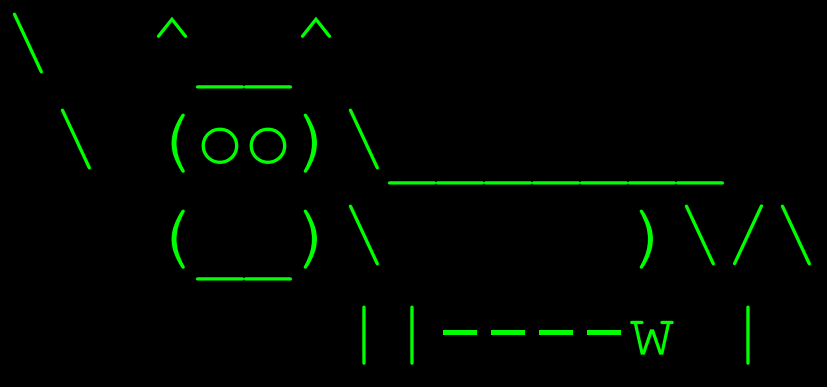
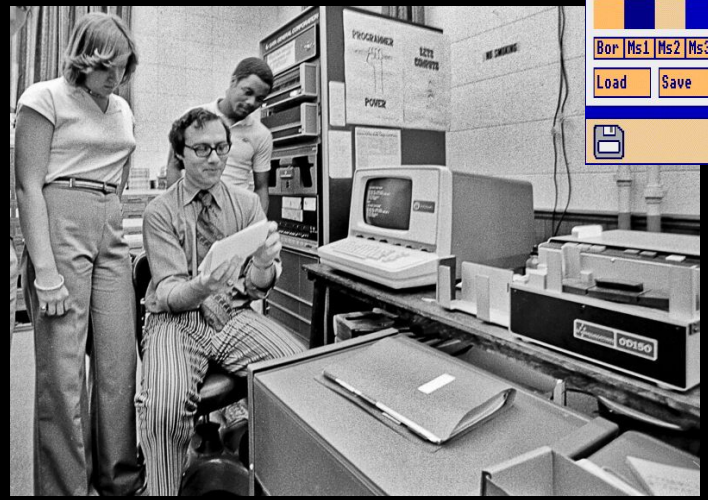
< Bash Shell for Researchers >

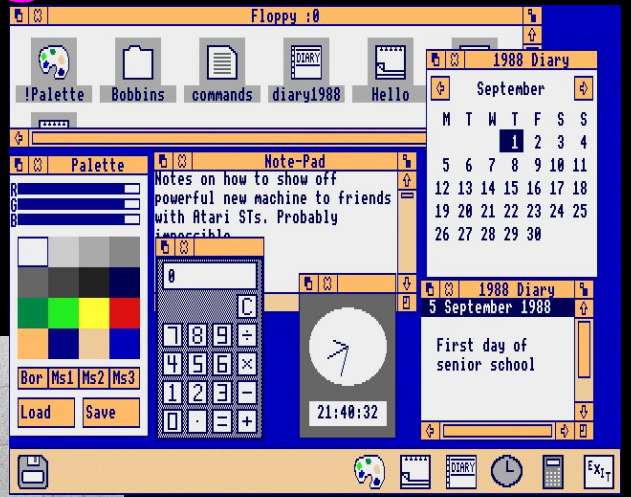


Dan Kerchner
GW Libraries & Academic Innovation
September 13, 2023

How you interact with your computer:

- Graphical User Interface
- Command Line





Why Command line? Why Bash?

- Many scientific computing tools can ***only*** be used through a command line interface
- Automate/accelerate repetitive tasks
- Make your work less error-prone and more reproducible
- Often the only way to access a server ("cloud computer")

  Reasons you need to work on a server and not your laptop:
  - It's "always on" and "always connected"
  - Bigger and faster than your laptop
  - That's where the data and/or tools are

Why Command line? Why Bash?

- High prevalence of Unix[-like] systems (ref)
  - 77% of public web servers (vs. 23% Windows)
  - 71% of smartphones/tablets (Android)
  - 38% of embedded systems
  - 29% of desktops/laptops (most are Mac)
  - 100% of supercomputers

- Critical literacy in your coding toolkit

You can interact with your:

- Mac
- Unix[-like] server
- Windows

devices

using Bash* shell

*or similar

Starting a Bash shell session

- **Locally**:
  - **Mac:** Terminal app
  - **Windows:** Windows Subsystem for Linux (WSL)* or Git Bash
    NOT: Command Prompt or PowerShell

*may require some configuration

- **Remote Unix-type server:**
  Locally, "ssh" to remote server

Shell Commands

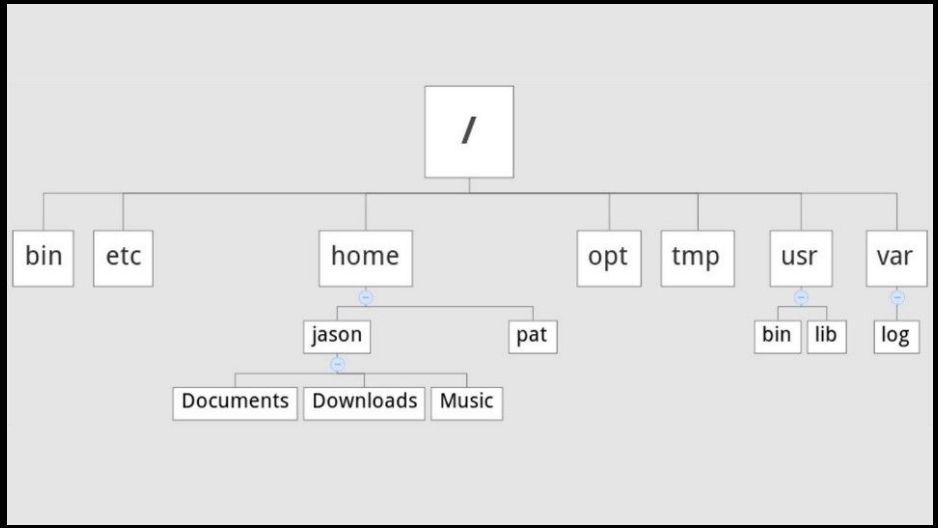
```
command argument argument -option
     -option value --option=value
```

Examples:

- `find /usr/share -name iris*`
- `cp -r project1/data project2/data`

Where am I? The file system

- Terminology:
  - Directories or Folders
  - Files
  - Links
  - Permissions



File system navigation

- `pwd   # (what's the) presentq working directory`
- `ls    # list contents (-l for "long")`
- `cd    # change directory`
- `mv    # move (-r for recursive)`
- `cp    # copy (-r for recursive)`
- `rm    # remove (-r for recursive)`
- `mkdir # make directory`
- `rmdir # remove directory`

File system navigation

```
/     # file system root
.     # this directory (right here)
..    # parent directory (one level up)
~     # my home directory
/absolute vs. relative paths
```

Filename hacks

Unix can be fairly permissive about characters - including spaces - in file names

- Using "" quotes can help
- \ (backslash) can be used to indicate "take the next character literally" (for example, if a file name contains a space)

Wildcards in file names

`*` Matches **0 or more** matching characters

`?` Matches **1** matching character

Examples:

```
Ex*s          # matches Examples, Exits, Exs...
myfile0?.txt  # matches myfile01.txt,
                      myfile0x.txt, ...
```

File content

- Editors: nano, vim
- Commands:
  - head, tail # preview the [first, last] few lines
  - more, less # scroll through file
  - cat # concatenate
  - wc # word/line count
  - sort # sort lines
  - gzip/gunzip # gnu zip/unzip

Permissions

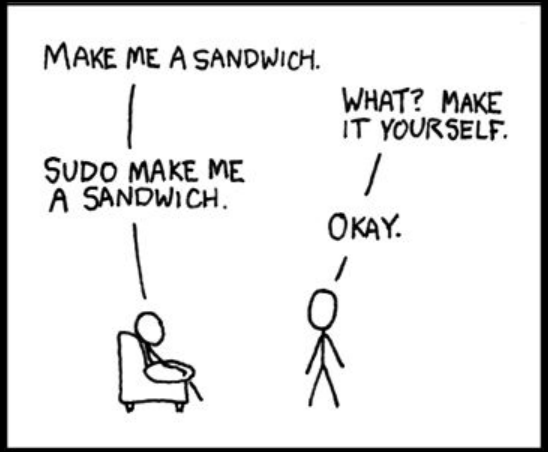
On Files AND on Directories

```
drwxrwxrwx
^          is a directory
 ^^^       read/write/execute for the owning user
    ^^^    read/write/execute for the owning group
       ^^^ read/write/execute for the world
```

Example:

`-rw-r--r--` I can read/write, all others can read

Permissions



Commands include:

- chmod # change mode
  # ex: chmod +x script1.sh
- chown # change owner
  # ex: chown dan:dan myfile.txt
- sudo # do as "su" (superuser)
  # you must be in the sudoers group
  # ex: sudo chown dan:dan myfile.txt

Who am I? Users and groups

- root is the superuser
- sudo su - *anotheruser* # be another user
- Usually maintained in /etc/group and with commands
- whoami # who am I?
- who # who else is logged on?

Getting help

- Many (not all) commands offer help via:
  - man *some_command* # manual page
  - *some_command* --help # often more brief
  - *some_command* -h

Downloading from the Internet

- `wget, curl`

Finding and searching

- `find`
- `grep`

Putting things together

```
>     # redirect output to a new file

>>    # redirect output, append to file

|     # pipe output into another command
```

Even more useful commands

```
echo    # print something out
history # view command history
```

Installing things: Package management

- `apt      # for most Linux distributions`
- `brew     # on Mac`
- `and others`

Scripting: DIY commands

- Make your own commands
  - Helpful for repeating/reproducing

Script: An example

```
#!/bin/bash
# Converts Tif/tif files to PDF via JPEG compression at 100% quality.
# If resulting PDF is >= 10MB, reconverts with 50% quality (and does not check size again)

# Check for 1 argument
if [ "$#" -ne 1 ]; then
  echo "Usage: $0 BINDERNUMBER" >&2
  echo "Example: $0 05" >&2
  exit 1
fi

for f in $(ls /Users/kerchner/Box/ISIS_Files_Renumbered/TIFFs/Binder_$1/$1_*.tif)
do
    echo -n "converting $f to ${f/tif*/pdf}..."
    convert -limit memory 0 -limit map 0 "$f" -compress jpeg -quality 100 "${f/tif*/pdf}"
    filesize=$(stat -f%z ${f/tif*/pdf})
    echo "...done"
    if ((filesize > "10000000"))
    then
        echo "reconverting $f to ${f/tif*/pdf} at 50%..."
        convert -limit memory 0 -limit map 0 "$f" -compress jpeg -quality 50 "${f/tif*/pdf}"
        echo "...done"
    fi
done
```

Want to learn more?

- Software Carpentry
  swcarpentry.github.io/shell-novice/
- man bash (manual page for "bash")
- linkedin.com/learning (log in w/GW credentials)

One-on-one help!

Coding consultations: go.gwu.edu/coding

(choose "other" for questions related to Bash/shell)

Dan Kerchner kerchner@gwu.edu