Student Mini Lessons

CS14 - S26

Topics

- proc/procinfo (max)
- dig (jaehoon)
- wget (melissa)
- curl (carson)
- rsync (eli)
- fzf (troy)
- strace (oscar)
- hexdump (wendy)
- time (jaron)
- tee (liam)

Guidelines

- Presentation should be 3-5 minutes
- Explain the purpose of the tool and, if applicable, why it is better to use this tool than one we already have learned
- Give multiple examples of using the tool with different options
- Include either a follow-along exercise or a short independent activity using the tool
- Prepare to answer questions about the tool and how it relates to other material we've learned this semester
- Add your slides to this presentation!

proc/procinfo (max)

Have you ever forgotten an important date? Maybe a loved one's phone number? Or maybe even your social security number? Well, you're in luck, because procinfo can help you with none of these things.

What procinfo **can** help with:

- Viewing system metrics
  - Memory consumption
  - System uptime
  - System interrupts

Helpful procinfo flags:

- n<insert number here>
- H
- h

Exercise:

- Run `procinfo` and find uptime
- Search for uptime in proc/
- Run `htop` and find uptime

Which of the following methods did you prefer for finding out information about your system?

(jaehoon)

dig - Domain Information Groper

ℹ️ What it is

A commandline that queries DNS servers and gives verbose data on how domain resolves to an IP address.

ℹ️ Why it's useful

- Gives full information on DNS
- Helps developers to debug domain related problems.
  - Setting up homepage
  - Email providers
  - Domain ownership

💡 How to use it

`dig [domain] .`
Basic Look up

`dig [domain] +short`
clean minimal output

`dig [domain] [types] .`
type: A, AAAA, MX, NS, TXT, CNAME

`dig @server [domain] .`
query to specific server

`dig -x [ip] .`
Reverse lookup ip -> hostname

💡 Examples

```
bash$ dig swarthmore.edu MX +short
10 alt3.aspmx.l.google.com.
1 aspmx.l.google.com.
10 alt4.aspmx.l.google.com.
5 alt2.aspmx.l.google.com.
5 alt1.aspmx.l.google.com.

bash$ dig swarthmore.edu NS +short
ns0213.secondary.cloudflare.com.
ns0085.secondary.cloudflare.com.

bash$ dig harvard.edu MX +short
100 mx0a-00171101.pphosted.com.
100 mx0b-00171101.pphosted.com.

bash$ dig college.harvard.edu MX +short
5 alt2.aspmx.l.google.com.
1 aspmx.l.google.com.
10 alt4.aspmx.l.google.com.
5 alt1.aspmx.l.google.com.
10 alt3.aspmx.l.google.com.
```

wget – World Wide Web Get (melissa)

`wget` allows users to retrieve files from the web using common internet protocols (i.e. HTTP, HTTPS)

Can be useful for slow, unstable connections, archiving, resuming downloads

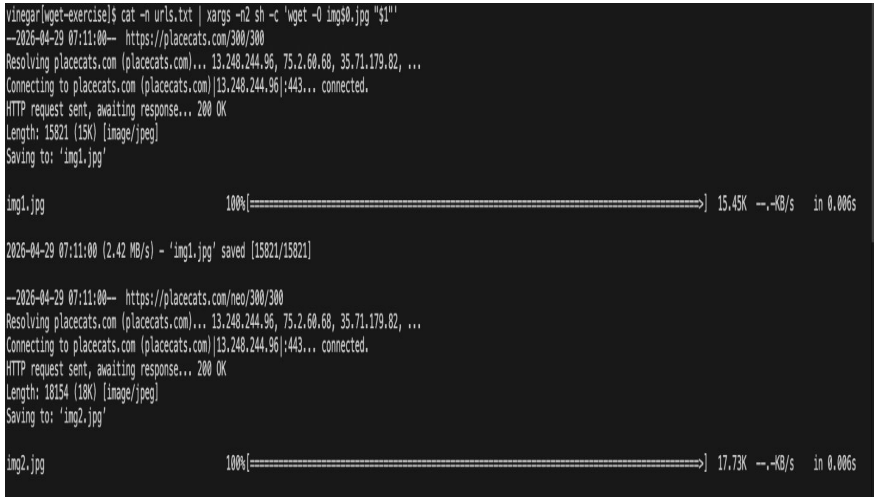
Common options:

- -c: Continue an interrupted download
- -O: Specify filename to save download to
- -m: Create a mirror copy of a website
- -i: Download multiple input files at once

Quick exercise: Run the following to fetch image files from the internet

`-cp /home/mmckenz2/public/cs14/wget-exercise`

`-./makeimg.sh`



Then run `firefox img[num].jpg` on your image of choice

curl: lets you talk to the internet from the command line

Carson

How to use it

Basic: `curl [url]`
Show only headers: `curl -I [url]`
Save output to a file: `curl -o [file] [url]`
Send post request: `curl -X POST [url]`
Send form data: `curl -d "name=James" [url]`
Custom headers: `curl -H "Content-Type: application/json" [url]`

Why it's useful

1. Tests websites and APIs quickly
2. Shows raw server responses
3. Helps debug HTTP requests
4. Can download files
5. Can send data like forms or JSON

Examples:

```
~
❯ curl https://example.com
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" conte
nt="width=device-width, initial-scale=1"><style>body{background:#eee;width:60vw;margin:15vh a
uto;font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{colo
r:#348}</style></head><body><div><h1>Example Domain</h1><p>This domain is for use in document
ation examples without needing permission. Avoid use in operations.</p><p><a href="https://ia
na.org/domains/example">Learn more</a></p></div></body></html>

~
❯ curl -I https://example.com
HTTP/2 200
date: Wed, 29 Apr 2026 03:06:48 GMT
content-type: text/html
server: cloudflare
last-modified: Sat, 18 Apr 2026 00:51:00 GMT
allow: GET, HEAD
accept-ranges: bytes
age: 9254
cf-cache-status: HIT
cf-ray: 9f3b17c6a930811f-EWR
```

Exercise:

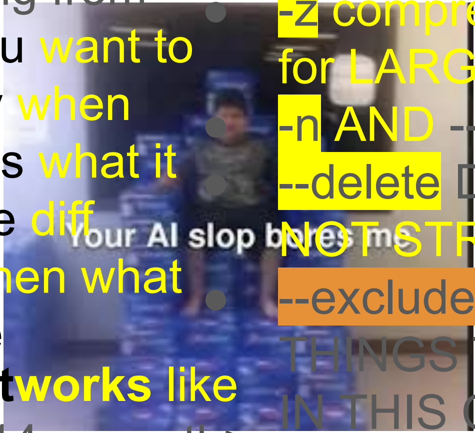
Try `curl http://40.160.231.143:8001`, then follow the clues in each server response until you unlock the secret endpoint.

Eli mogger **USEGE** rsync <source_path> <dst_path>

rsync is just cp but also scp and more efficient 

it's basically a git pull command but without github needed and you can do it without being in the directory you're pulling from

- You should use this tool if you want to copy things faster, especially when syncing files (as it only copies what it needs to in order to make the diff between two places 0), or when what you are copying is very large



It can be used **locally**/**across networks** like

scp -r <cslab_path> cs14vm:<cs14vm_path>

Aight now I want y'all to copy your ENTIRE CS14 directory AND YOUR ENTIRE CS35 DIRECTORY into your CS14VM using THE FLAGS -a, -v, and -z 😈

FLAGS

- -a preserves the source state. So permissions, timestamps, symlinks. It also recursively copies directories
- -v logs everything transferred
- -z compresses what's transferred (good for LARGE FILES)
- -n AND --dry-run simulate the copy
- --delete DELETES EVERYTHING THATS NOT STRICTLY IN THE SOURCE! :0
- --exclude='*.log' would NOT COPY THINGS THAT MATCH THE PATTERN, IN THIS CASE LOG FILES
- NOTE: **rsync -a /src /dst/** copies the source directory, while **/src/** only copies whats within it!!!

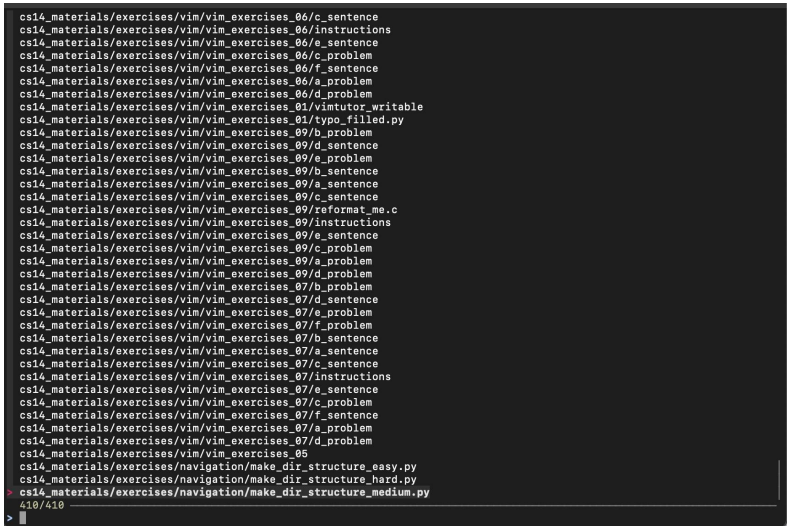
fzf (troy)

Codename: Fuzzy Finder

Fzf is a command that operates as a separate interface. Inside the interface, it works like an open search, showing a list of all files and directories accessible from the your current path.

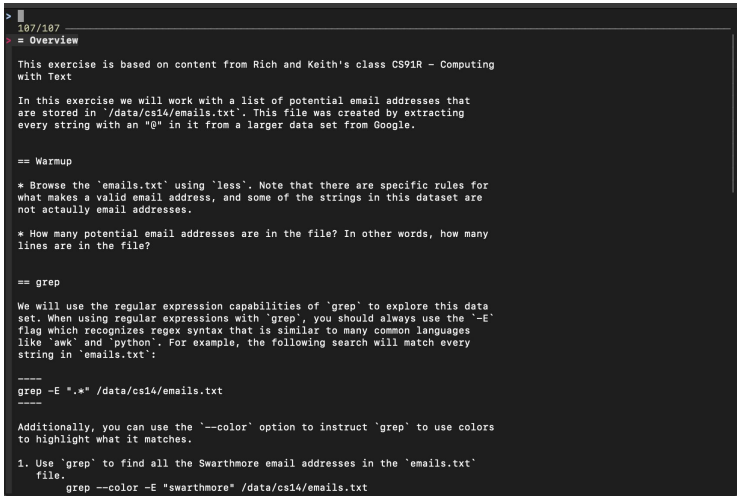
fzf [options]
 –reverse (reverses the order of search output)
–m (allows you to use tab and shift+tab to select multiple files)
–no-sort (does not sort result)

‘Enter to confirm and print your selection’

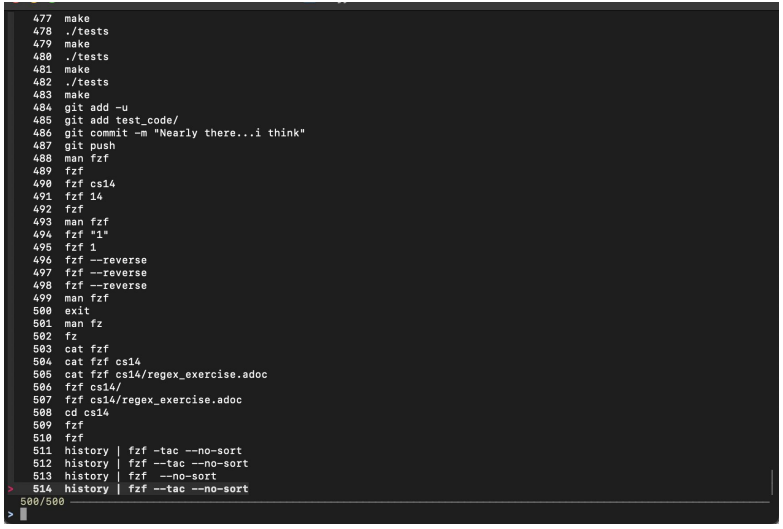


Fzf examples

cat regex_exercise.adoc | fzf –reverse



fzf applied to nothing will only interact with search list of files and directories. To interact with another specific command or within a file, you would have to pipe it



history | fzf –tac –no-sort

Strace (Oscar)

Strace

Strace (short for “system call trace”) interprets and logs every system call a process makes. This lets you see exactly how your program interacts with the kernel.

Strace shows every system call your program makes in real time, so when something fails or hangs, you can see exactly what it is asking the OS to do. Strace does this with no source code or recompilation needed.

Why Strace

Strace isn’t meant to be used in place of other debugging methods, it fills in where they fall short.

- Print statements only show what the developer thought to log. If the bug is in a library or system call for instance, you get nothing
- Gdb requires source code and can be heavy to set up on a live system

Strace works on any binary, no source needed, no recompilation, no instrumentation. You attach and immediately see every interaction with the OS.

Reading the output

openat(AT_FDCWD, “/etc/hosts”,0_RDONLY)=3

Syscall name arguments return value

The syscall name is the kernel function being called (read, write, open, execute)

Arguments are inputs passed to the syscall (file descriptors, paths, flags, buffers)

Return value, ≥0 means success (often fd or byte count); -1 means error

Useful Flags

-e trace=file filters output to only file-related syscalls (open, read, stat…)

-e trace=network filters output to only network-related syscalls (connect, bind…)

-p <pid> attaches to an already-running process by PID

-f follows child processes (forks and threads as well)

-o <output.txt> saves output to a file instead of stderr

-c prints a summary count/time per syscall at the end

- -tips shows a little dude to give you a random tip on strace

Exercise

A hostname is the name your machine goes by on a network. You might look it up when configuring SSH, setting up a server, or just confirming which machine one is on after logging into a remote system.

You can use strace to try to find your device’s hostname!

Hint: It is found in the directory /etc/

Example (Answer)

RUN: $strace -e trace=file cat /etc/hostname

Find the successful open: openat(AT_FDCWD, “/etc/hostname”, O_RDONLY)=3

Spot the read: read(3, “m yhost\n”, 131072)=7

This is your hostname!!

Thank you!

hexdump (wendy)

Outputs files in hexadecimal, decimal, octal, or ascii

Useful for reconstructing files from their binary form & debugging

For (more) human readable output:

- hexdump -C <file> **Output rows:** offset hexadecimal |ascii|
- hexdump -c <file> **Output rows:** offset 16 space separated ascii chars

Exercise: try reconstructing the given file

/home/ckazer/public/cs14/inclass/secret_bash

```
0749eb90  f0 32 7d 60 95 48 d0 62  08 80 4b 67 b4 4a 21 dc  |.2}`.H.b..Kg.J!.|
0749eba0  80 3f 6c dd 4a f5 a3 d4  ce 32 8d e4 21 d7 a5 5a  |.?l.J....2..!..Z|
0749ebb0  92 93 4b f1 ca 0a ce 3c  b9 14 20 a5 00 a4 4a 3e  |..K....<.. ...J>|
0749ebc0  bd 4b 8c b4 d1 90 2b 25  a9 c8 f4 c8 10 85 fb d6  |.K....+%........|
0749ebd0  fc 2a 1f c6 8a 7f 25 e7  47 f4 95 01 e2 d7 82 fe  |.*....%.G.......|
0749ebe0  22 95 fa 8e 49 e4 50 98  d3 84 95 a7 97 1d 97 92  |"...I.P.........|
0749ebf0  25 32 9f 90 0c a9 07 73  c2 2b 49 06 4c 1a 26 69  |%2.....s.+I.L.&i|
0749ec00  b2 75 3e 20 db 65 bf 22  68 cf 29 1b 8a 65 8d 54  |.u> .e."h.)..e.T|
0749ec10  91 ba 33 f3 05 59 07 39  cd 43 96 6f 5d 88 bb 7a  |..3..Y.9.C.o]..z|
0749ec20  aa ae d2 04 b1 c6 33 25  8c 68 f7 c7 79 23 ef 66  |......3%.h..y#.f|
0749ec30  7a aa 41 e7 99 55 1d 46  79 64 2a 6c 1f a9 64 63  |z.A..U.Fyd*l..dc|
0749ec40  ef f9 87 72 3f d9 5a 9f  48 0d 92 96 72 0d 1b a4  |...r?.Z.H...r...|
0749ec50  a6 2e 08 b0 96 cc e6 37  88 f0 57 32 3b 21 6d d9  |.......7..W2;!m.|
0749ec60  e4 6b f1 ef 14 25 65 e3  3c b3 ee 60 bc a4 ea 44  |.k...%e.<..`...D|
0749ec70  64 49 0d 59 0b 45 3f f0  75 a4 24 be 41 f5 52 ad  |dI.Y.E?.u.$.A.R.|
0749ec80  32 65 33 4d 9c 83 8e 97  69 57 f2 5d 72 93 dd b1  |2e3M....iW.]r...|
0749ec90  d0 c6 dc c8 43 89 6e 1e  8b d9 2e 67 52 3e 26 3f  |....C.n....gR>&?|
0749eca0  46 cc 92 a7 e1 f3 af 9c  c8 b3 17 fe ff 8a bb 7a  |F..............z|
0749ecb0  f6 e9 99 6d 8b 24 dc 84  97 67 b6 d5 5b 73 a6 fc  |...m.$...g..[s..|
0749ecc0  50 a6 cf fe 92 7d c3 2f  2e 7e e8 b7 8f 9b 71 5f  |P....}./.~....q_|
0749ecd0  b0 43 79 5c f1 63 9d b7  2f 7e b1 f3 f6 87 5f b0  |.Cy\.c../~...._.|
0749ece0  64 84 86 98 59 f7 d2 96  42 28 5a 96 8e d1 17 4f  |d...Y...B(Z....O|
0749ecf0  f4 2d a6 94 06 0f fb 57  83 fe 60 59 8e 32 70 23  |.-.....W..`Y.2p#|
0749ed00  c1 8a 98 43 0b 90 26 24  03 ce 3d 21 79 0b 75 f9  |...C..&$..=!y.u.|
```

April 29, 2026

TIME

THE COMMAND THAT CAPTURES TIME ITSELF

Used to gauge the time or resource usage of a command.

```
time [option...]
command [argument...]
```

HOW DO WE MEASURE TIME?

ELAPSED REAL TIME - real time that has passed

USER CPU SECONDS - the time it took for the processor to crunch the numbers

SYSTEM CPU SECONDS - the time it took for the OS to help out using system calls

PLENTY OF OPTIONS

```
-o FILE
-a
-f FORMAT
-p
-v
```

`tee` -- duplicate standard input (liam)

`tee` duplicates stdin to stdout; it has the side effect of writing a copy of stdin to its arguments.

From the POSIX Programmer's Manual:

> The `tee` utility shall copy standard input to standard output, making a copy in zero or more files. The `tee` utility shall not buffer output.
>
> The following options shall be supported:
>
> `-a` Append the output to the files.
>
> `-i` Ignore the SIGINT signal.

tee -- duplicate standard input

`tee` is very useful when you wish to log an interactive program.

Choose an interactive shell (`sh`,`bash`,`python3`,...).

```
$ mkdir tee_time && cd $_
$ <your_cmd> <args> | tee <your_cmd>.log
```

Do something in your shell, then print `<your_cmd>.log`:

```
$ less -F <your_cmd>.log
```

You should see all standard output duplicated in your log file.

tee -- duplicate standard input

Recall that in unix, streams are files; `tee` can write to named streams

In a screen or tmux session:

```
$ mkfifo tmpstr                 ## create a named stream
$ cat tmpstr | tee              ## continuously print tmp
```

In another terminal, write something to `tmpstr` (i.e. `` `fortune > tmpstr` ``):

Use `` `cat tmpstr` `` to flush the stream

tee -- duplicate standard input

One might write `log_cmd`, which execute a command and duplicates the output between stdout and appending to the command name.

```
#!/bin/sh
command $* | tee -a $(basename "$1").log
```

Along with your (or someone else's) utility, where might you use `tee`?

See also:

`sponge` – soak up standard input and write to a file