Crash Course on UNIX and Systems Tools

Day 2 --- Scripting

Building: *Copying (forgot this yesterday!)*

`cp`

- "Copy" --- by specifying an input file and an output file
    - **`-R`** for **recursive** copying (useful for directories)
    - **`-a`** for "**archiving**" (*preferred flag*, matching all file attributes and content)

```
$ mkdir -p outer/inner ; touch outer/inner/test
$ cp -a outer copied_outer
```

Overview

- Part 1
  - Exercises
  - Scripting
  - Text Editors
- Part 2 (maybe today … maybe tomorrow)
  - So you have some code …
  - Version control (git)
  - Makefiles
  - Compilation

Exercises: *Questions*

1. Does your sandboxed machine support “**`avx`**” instructions (according to **`lscpu`**)?
2. How many active sockets (according to **`netstat`**) are of type “**`DGRAM`**”?
3. Can you check how many sockets (according to **`netstat`**) are of type **`DGRAM`** *every 5 seconds*? ***

Exercises: *#1*

- `lscpu` has some **output** that we could as **input** for a **pattern match**

```
$ lscpu | grep "avx"
```

Exercises: *#2*

- `netstat`'s output shows each active socket (and type) **per line**
- We could use the **output** as **input** for a **pattern match**
- We could then **count** the **number of lines** that matched

----

```
$ netstat | grep "DGRAM" | wc -l
```

(What's `-l` ? See the manual!)

Exercises: *#3*

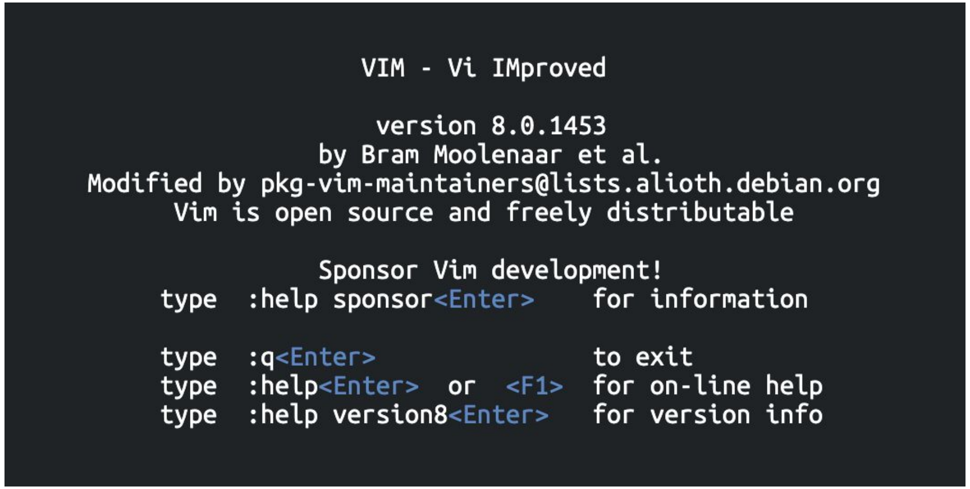
Would be nice to check in **a loop,** maybe use a **script**, and **view** the output **incrementally** ...

Scripting: *Basics*

- A script --- some code or a grouping of commands that are typically intermediately **interpreted** and run
    - Contrary to an executable binary (executing specific instruction sequences on a machine)
- We'll be writing some scripts in **bash**

Scripting: *Text editors*

- We probably want to write scripts using a better environment than **`cat`** or **`echo`** --- command line offers **text editors**!
- We'll be using **`vim`**



```
                     VIM - Vi IMproved

                      version 8.0.1453
                  by Bram Moolenaar et al.
    Modified by pkg-vim-maintainers@lists.alioth.debian.org
          Vim is open source and freely distributable

                  Sponsor Vim development!
       type  :help sponsor<Enter>    for information

       type  :q<Enter>               to exit
       type  :help<Enter>  or  <F1>  for on-line help
       type  :help version8<Enter>   for version info
```

Scripting: *Text editors*

- We’ll cover common shortcuts, keywords, and commands --- but a more comprehensive cheat sheet can be found here: **https://www.keycdn.com/blog/vim-commands** (preferred) and https://vimsheet.com/ . (See **Day 3** for **`vim`** configurations)

```
$ vim myscript
```

- **`esc`** + “**`:q`**” to quit!

Scripting: *Getting started*

- A script should have specify the correct **interpreter** to the OS
- The interpreter should be specified at the top as follows:

```
#!/bin/bash
```

- How do we know this?

Scripting: *Getting started*

`which`

- Outputs the location of a program or executable with respect to the **`PATH` environment variable**
- **`PATH`** is a list of directories where common programs exist
    - Each directory separated with the "`:`" token
- The OS looks in **`PATH`** to find common programs to execute

Scripting: *Getting started*

```
$ echo $PATH
...  (left out)

$ which bash
/bin/bash
```

Scripting: *Tackling the problem*

Back to the **question** ---- Can you check how many sockets (according to **`netstat`**) are of type **`DGRAM`** *every 5 seconds*? ***

We said --- “would be nice to check in **a loop,** maybe use a **script**, and **view** the output **incrementally** …”

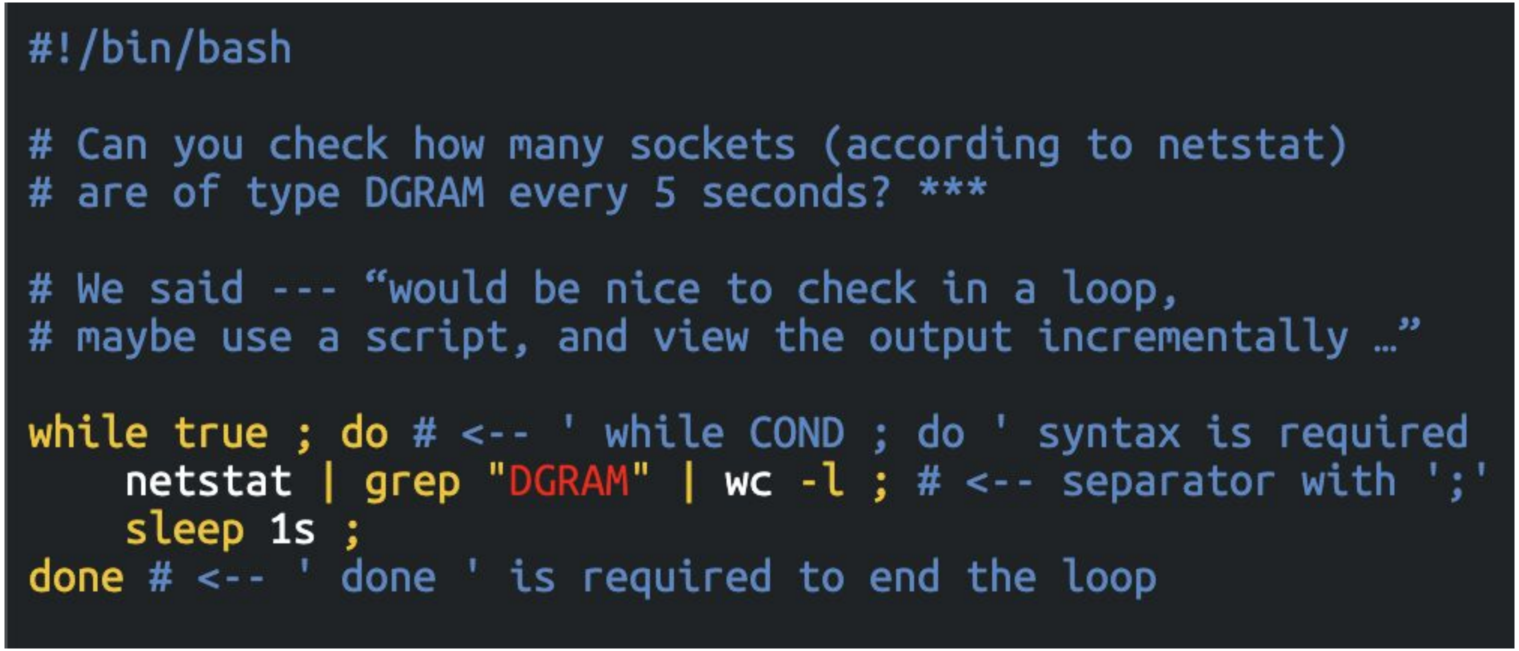
Scripting: *First script*

NOTE

- **All** code/scripts will be **documented** and **uploaded** to the Course Drive
- This includes a very **pedantic** version of the script (with **in-depth** explanations of bash syntax and techniques, etc.)

Scripting: *First script*

`$ vim first`



```
#!/bin/bash

# Can you check how many sockets (according to netstat)
# are of type DGRAM every 5 seconds? ***

# We said --- “would be nice to check in a loop,
# maybe use a script, and view the output incrementally …”

while true ; do # <-- ' while COND ; do ' syntax is required
    netstat | grep "DGRAM" | wc -l ; # <-- separator with ';'
    sleep 1s ;
done # <-- ' done ' is required to end the loop
```

Scripting: *Executing the script*

- The script is just a simple plain-text file … **how do we run it**?

```
----------

$ first
Command 'first' not found
```

- System tries to find the script in **`PATH`**, but fails

Scripting: *Executing the script*

- Alternative --- specify the directory in which the script is located to prevent a search into environment variables

```
----------

$ ./first
bash: ./first: Permission denied
```

- Permissions ... remember `ls -l` ?

Scripting: *Executing the script*

```
$ ls -l
…
-rw-r--r-- 1 root root  421 Jan  5 09:51 first
```

- Recall that permissions can be read (**`r`**), write (**`w`**), and execute (**`x`**)
  - For “first,” there are no executable permissions

Scripting: *Executing the script*

`chmod`

- “Change file mode” --- Can change permissions for a file
- Supply the mode “bits” and the file as parameters

```
----------

$ chmod +x first ; ls -l
…
-rwxr-xr-x 1 root root  421 Jan  5 09:51 first*
```

Scripting: *Managing the running process*

`ctrl-z + bg`

- **`ctrl-z`** **pauses** the current process
- **`bg`** places the process in the **background**, where it will run
  - Can optionally specify a job “ID” from the **`jobs`** command
- Optionally, you can start a process in the background directly by adding “**`&`**” at the end of your command

Scripting: *Managing the running process*

```
$ ./first
^Z
[1]+  Stopped                      ./first
$ jobs
[1]+  Stopped                      ./first
$ bg
[1]+ ./first &
$
```

Scripting: *Managing the running process*

`fg`

- How do we bring back the background process? “Foreground”
  - Can also optionally specify a job “ID” from **`jobs`**

```
----------

$ fg
./first
```

Scripting: *Managing the running process*

Unfortunately our script keeps outputting values every 5 seconds to the shell ... how come? **`stdout`** and **`stderr`** still appears on the screen!

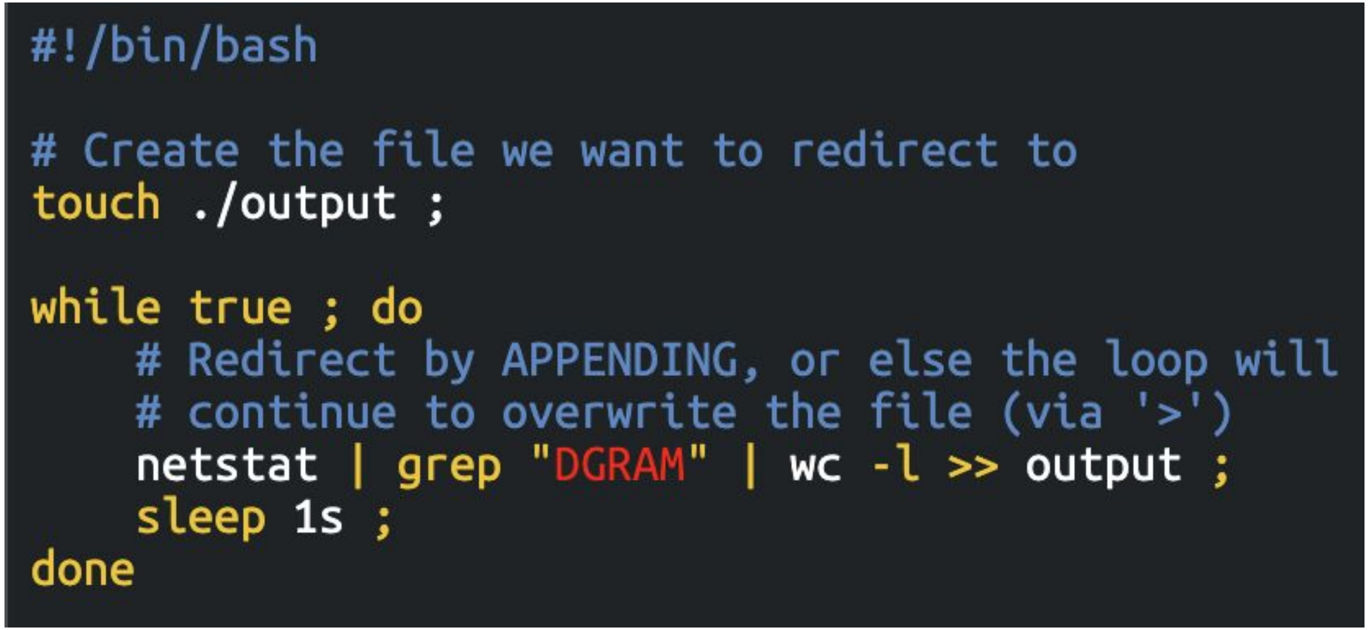
Solution --- redirection!

```
$ ./first > first.out &
[1] 2726
```

--- job ID, process ID (See using **`ps`** command)

Scripting: *Enhancing the script*

- Output to a file by default

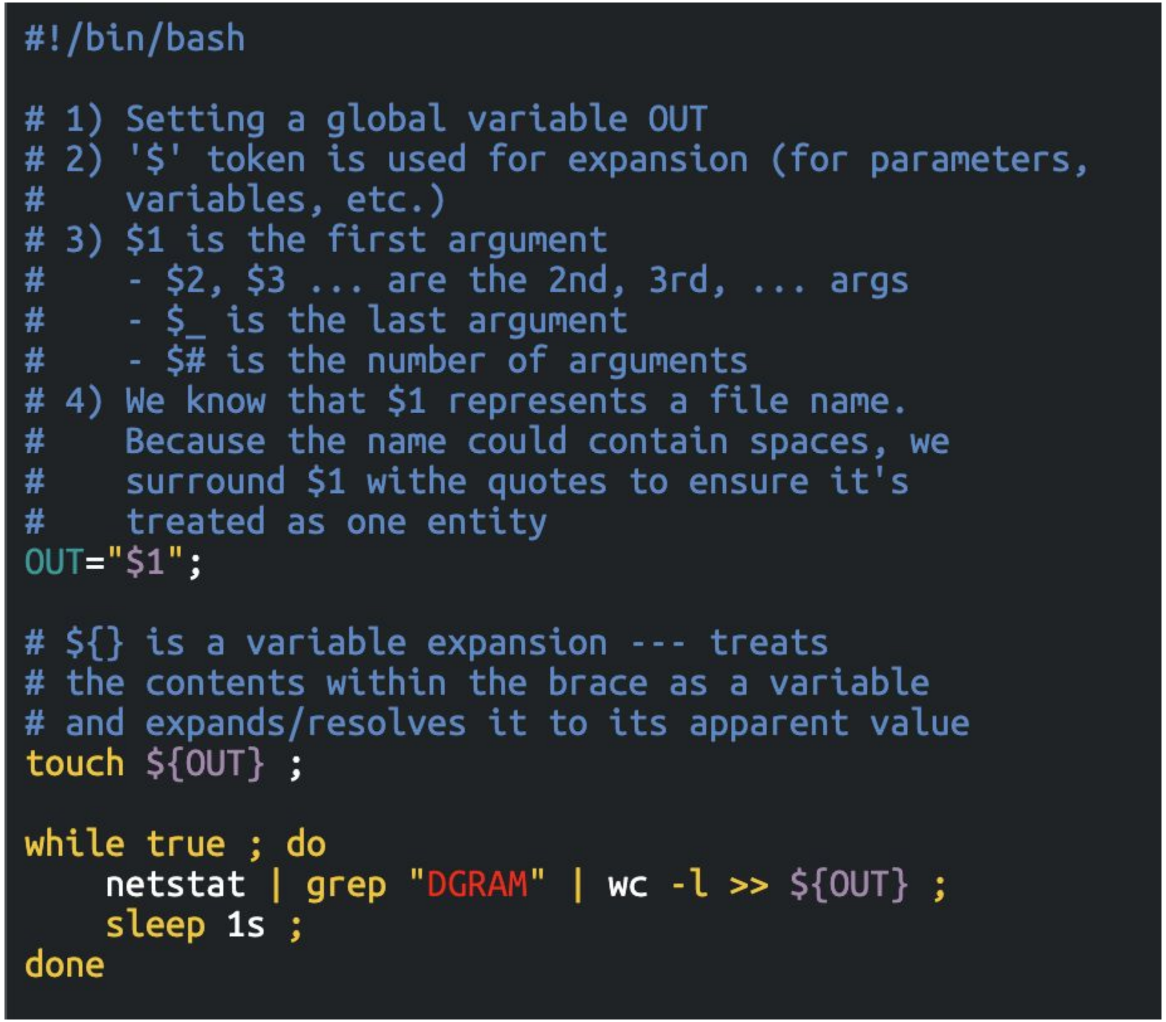


```
#!/bin/bash

# Create the file we want to redirect to
touch ./output ;

while true ; do
    # Redirect by APPENDING, or else the loop will
    # continue to overwrite the file (via '>')
    netstat | grep "DGRAM" | wc -l >> output ;
    sleep 1s ;
done
```

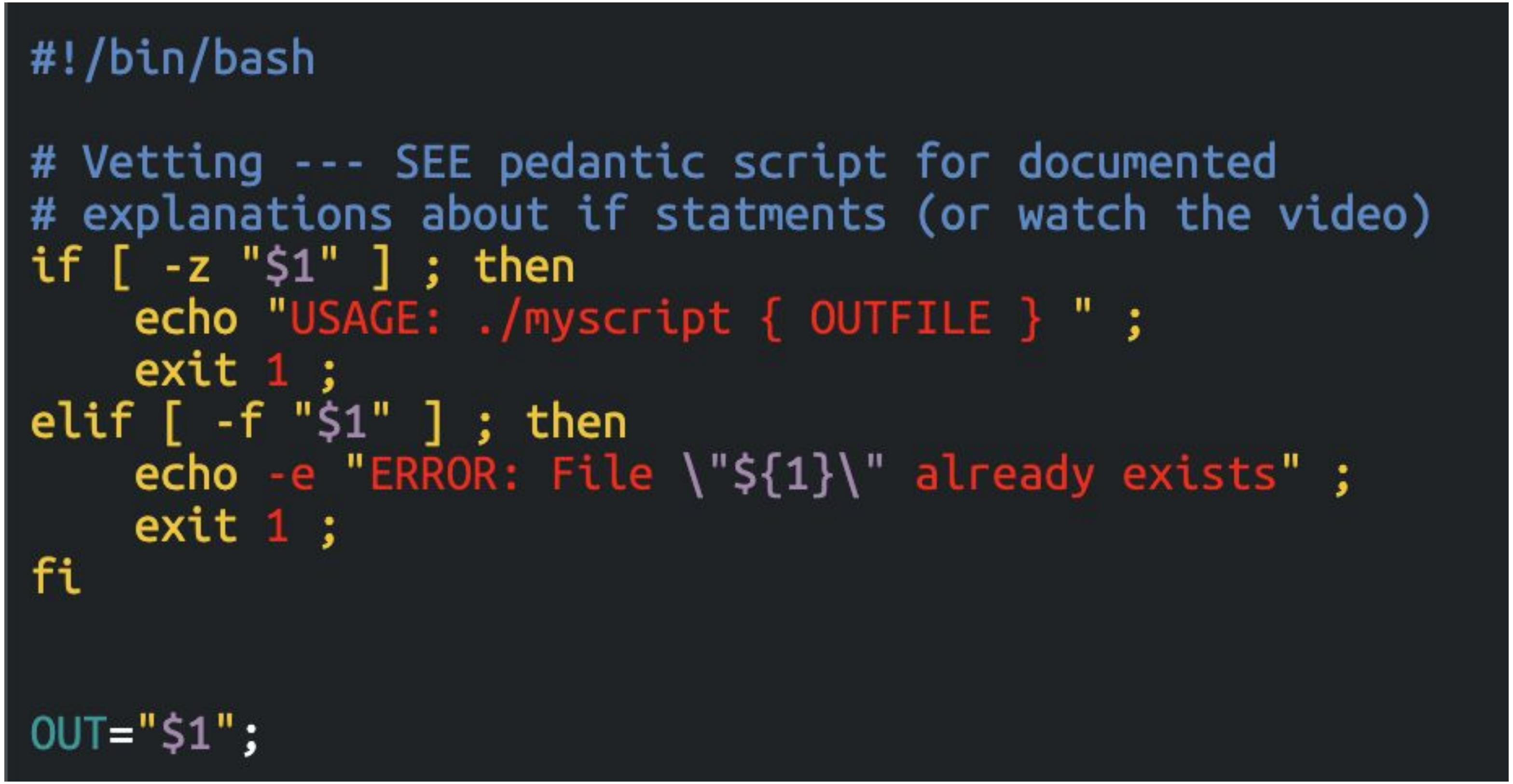
Scripting: *Enhancing the script*

- Allowing **user input** --- specify the filename
- **Vet**ting the input
  - Was there an actual input?
  - Should the specified file be overridden if it already exists?
- Providing **feedback** to the user

NOTE --- The slides will only cover a few concepts --- please see the **pedantic** script on the Course Drive for documented examples.



```bash
#!/bin/bash

# 1) Setting a global variable OUT
# 2) '$' token is used for expansion (for parameters,
#    variables, etc.)
# 3) $1 is the first argument
#    - $2, $3 ... are the 2nd, 3rd, ... args
#    - $_ is the last argument
#    - $# is the number of arguments
# 4) We know that $1 represents a file name.
#    Because the name could contain spaces, we
#    surround $1 withe quotes to ensure it's
#    treated as one entity
OUT="$1";

# ${} is a variable expansion --- treats
# the contents within the brace as a variable
# and expands/resolves it to its apparent value
touch ${OUT} ;

while true ; do
    netstat | grep "DGRAM" | wc -l >> ${OUT} ;
    sleep 1s ;
done
```



```bash
#!/bin/bash

# Vetting --- SEE pedantic script for documented
# explanations about if statments (or watch the video)
if [ -z "$1" ] ; then
    echo "USAGE: ./myscript { OUTFILE } " ;
    exit 1 ;
elif [ -f "$1" ] ; then
    echo -e "ERROR: File \"${1}\" already exists" ;
    exit 1 ;
fi


OUT="$1";
```

Scripting: *Enhancing the script*

Picking this up for Day 3 …