ECE590
Computer and Information Security

Fall 2018

Shell Proficiency and Data Manipulation

Tyler Bletsch

Duke University

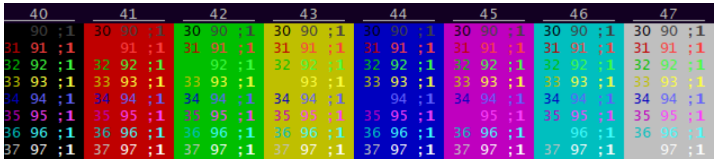
Motivation

- Everyone needs to manipulate data!
- Attackers need to:
  - Scan target environment for assets
  - Catalog and search target assets for possible vulnerabilities
  - Inspect binaries for specific instruction patterns
  - Extract specific data for processing by other tools (e.g. extracting password hashes from a user database)
- Defenders need to:
  - Scan own environment for assets *and* malicious entities
  - Catalog own inventory and compare against known vulnerabilities
  - Inspect traffic and data for known attack signatures
  - Extract specific data for processing by other tools (e.g. summarizing login failures to update a firewall blacklist)

Fundamental approach: UNIX Philosophy

- Combine simple tools to get complex effects
- Each tool does one thing and does it well
- Basic format of information is *always* a byte stream and *usually* text
- Core ingredients:
  - Shell (e.g. bash)
  - Pipes and IO redirection
  - A selection of standard tools
- Bonus ingredients:
  - SSH trickery
  - Regular expressions (HUGE!)
  - Terminal magic (color and cursor control)
  - Spreadsheet integration
  - More...

```
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
```



```
   40        41        42        43        44        45        46        47
   90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1
31 91 ;1     91 ;1  31 91 ;1  31 91 ;1  31 91 ;1  31 91 ;1  31 91 ;1  31 91 ;1
32 92 ;1  32 92 ;1     92 ;1  32 92 ;1  32 92 ;1  32 92 ;1  32 92 ;1  32 92 ;1
33 93 ;1  33 93 ;1  33 93 ;1     93 ;1  33 93 ;1  33 93 ;1  33 93 ;1  33 93 ;1
34 94 ;1  34 94 ;1  34 94 ;1  34 94 ;1     94 ;1  34 94 ;1  34 94 ;1  34 94 ;1
35 95 ;1  35 95 ;1  35 95 ;1  35 95 ;1  35 95 ;1     95 ;1  35 95 ;1  35 95 ;1
36 96 ;1  36 96 ;1  36 96 ;1  36 96 ;1  36 96 ;1  36 96 ;1     96 ;1  36 96 ;1
37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1     97 ;1
```

The bash shell and common Unix tools

The bash shell

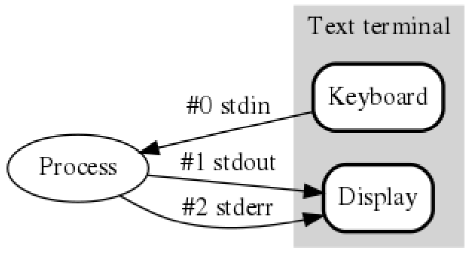
- Shell: Most modern Linux systems use **bash** by default, others exist
  - We’ll use bash in our examples

- Side-note: You can get a proper UNIX shell on Windows using Cygwin, MinGW, or other similar tools.
  - There’s also “Windows Subsystem for Linux”, but I don’t trust that thing yet...
  - PowerShell is Microsoft’s answer to bash...it’s fine.

Shell basics review

- **Standard IO**: stdin, stdout, stderr
- **Pipes**: direct stdout of one to stdin of another

```
ls | sort -r                # sort files reverse order
```



- **File redirection**: direct any stream to/from a file

```
ls > file_list.txt              # save ls to a file (note: no columns!)
gzip -c < archive.gz | wc -c    # how big is this file uncompressed?
find -iname dog.* 2> /dev/null  # supress stderr
```

- **Tab completion**: ALWAYS BE MASHING TAB!!!!!!!!!!
  - Once = complete, twice = list.

- **Semicolon** for multiple commands on one line

```
make ; ./myapp
```

- Can use && and || for short-circuit logic

```
make && ./myapp
```

(Based on **return value** of program, where 0 is success and nonzero is error)

Stuff from Homework 0 that I assume you know

- echo
- cat
- head
- tail
- less
- grep
- diff
- wc
- sort
- find
- sed
- awk

Note: The guy who did the Lynda video, Scott Simpson, has more videos. See Learning Bash Scripting for examples of some of the stuff in this lecture.

Bash syntax

- Expansions:
  - **Tilde (~)** is replaced by your home directory (try "`echo ~`").
    `~frank` expands to *frank*'s home directory.
  - **Braces** expand a set of options: `{a,b}{01..03}` expands into 6 arguments:
    `a01 a02 a03 b01 b02 b03`
  - **Wildcards**: `?` matches one char in a filename, `*` matches many chars,
    `[qwe0-3]` matches just the chars q, w, e, 0, 1, 2, or 3.
    - Non-trivial uses! Find all Makefiles two dirs lower: `*/*/Makefile`
  - **Variables** are set with `NAME=VALUE`. Values are retrieved with `$NAME`.
    Names usually uppercase. Fancy expansions exist, e.g. `${FILENAME%.*}` will get filename extension; see here for info. Variables can be made into *environment variables* with *export*, e.g. `export NAME=VALUE`.
- Quotes:
  - By default, each space-delimited token is a separate argument
    (different argv[] elements)
  - To include whitespace in a single argument, quote it.
    - Use **single quotes** to disable ALL expansion listed above
    - Use **double quotes** to allow variable expansion only

Bash syntax (2)

- Control and subshells

```
for NAME in WORDS... ; do COMMANDS; done
```

  - Execute commands for each member in a list.

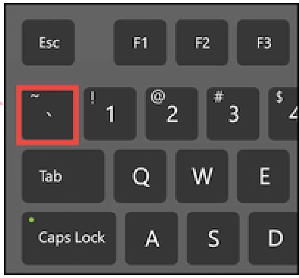
```
while COMMANDS; do COMMANDS; done
```

  - Execute commands as long as a test succeeds.

```
if COMMANDS; then COMMANDS;
[ elif COMMANDS; then COMMANDS; ]...
[ else COMMANDS; ] fi
```

  - Execute commands based on conditional.

`` `COMMAND` `` *or* `$(COMMAND)`



  - Evaluate to the stdout of COMMAND, e.g.:
    ``USERNAME=`whoami` ``

Control flow examples

- Keep pinging a server called ‘peridot’ and echo a message if it fails to ping.

```
while ping -c 1 peridot > /dev/null ; do sleep 1 ;
 done ; echo "Server is down!"
```

(Can invert by prepending ‘!’ to ping – waits for server to come *up* instead)

- Check to see if our servers have been assigned IPs in DNS:

```
for A in esa{00..06}.egr.duke.edu ; do host $A ; done
esa00.egr.duke.edu has address 10.148.54.3
esa01.egr.duke.edu has address 10.148.54.20
esa02.egr.duke.edu has address 10.148.54.27
esa03.egr.duke.edu has address 10.148.54.28
esa04.egr.duke.edu has address 10.148.54.29
esa05.egr.duke.edu has address 10.236.67.31
esa05.egr.duke.edu has address 10.148.54.30
esa06.egr.duke.edu has address 10.148.54.31
```

This stuff isn’t just for scripts – you can do it straight on the command line!

Conditionals: `[ ]`, `[[ ]]`, `(( ))`, `( )`

- Conditionals
  - **Commands**: Every command is a conditional based on its **exit status**
  - **Test conditionals**: Boolean syntax enclosed in *spaced-out* braces

`[ STR1 == STR2 ]`	String compare (may need to quote)
`[ -e FILE ]`	File exists
`[ -d FILE ]`	File exists and is a directory
`[ -x FILE ]`	File exists and is executable
`[ ! EXPR ]`	Negate condition described in EXPR
`[ EX1 -a EX2 ]`	AND the two expressions
`[ EX1 -o EX2 ]`	OR the two expressions

- See here for full list
- Double brackets get you newer bash-only tests like regular expressions:
  `[[ $VAR =~ ^https?:// ]]` VAR starts off like an HTTP/HTTPS URL
- Double parens get you arithmetic:
  `(( $VAR < 50 ))` VAR is less than 50
- Single parens get you a subshell (various sometimes-useful side effects)

What is a script?

- **Normal executable**: binary file in an OS-defined format (e.g. Linux “ELF” format) appropriate for loading machine code, marked with **+x** permission.
- **Script**: Specially formatted text file marked with **+x** permission. Starts with a “hashbang” or “shebang”, then the name of binary that can interpret it, e.g.:

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

or

```
#!/usr/bin/python
```

  - On execution, OS runs given binary with script as an argument, then any given command-line arguments. No shebang? Defaults to running with bash.
  - Example: “**`./myscript -a 5`**” is run as “**`bash ./myscript -a 5`**”.
  - Can also just run a script with bash manually (e.g. “**`bash myscript`**”)
- **When should you write a bash script?**
  - When the thing your doing is >80% shell commands with a bit of logic
  - Need lots of logic, math, arrays, etc.? Python or similar is usually better.

Examples (1)

- Making an assignment kit for another of my classes:

```
$ echo `ls` > buildkit
$ cat buildkit
Autograder_rubric.docx Autograder_rubric.pdf byseven.s
grading_tests homework2-grading.tgz
HoopStats.s HoopStats.s-cai hw2grade.py
HW2_GRADING_VERSION Makefile recurse.s
```

Dump all the filenames into the would-be script. The echo/backtick makes them space-delimited instead of newline-delimited.

```
$ nano buildkit
$ cat buildkit
tar czf kit.gz Autograder_rubric.pdf byseven.s
grading_tests hw2grade.py HW2_GRADING_VERSION Makefile
recurse.s
```

Edit it to add tar command and strip out stuff I don’t want to include.

```
$ chmod +x buildkit
$ ./buildkit
$ ls -l kit.gz
-rw-r--r-- 1 tkb13 tkb13 771264 Sep 14 18:14 kit.gz
```

Mark executable, run, verify tarball was created

Examples (2)

- Script to run the ECE650 “hot potato” project for grading:

```
#!/bin/bash
./ringmaster 51015 40 100 |& tee out-14-rm.log &
./player `hostname` 51015 |& tee out-14-p00.log &
./player `hostname` 51015 |& tee out-14-p01.log &
./player `hostname` 51015 |& tee out-14-p02.log &
./player `hostname` 51015 |& tee out-14-p06.log &
./player `hostname` 51015 |& tee out-14-p07.log &
./player `hostname` 51015 |& tee out-14-p08.log &
./player `hostname` 51015 |& tee out-14-p09.log &
wait
```

Backticks to get external hostname

Backgrounded

Pause until all child processes have exited.

Shorthand for “stdout and stderr together”

More common commands (1)

- **diff**: Compare two files
  - Example use: How does this config file differ from the known-good backup?

```
$ diff config config-backup
2d1
< evil=true
```

Second line, first column

Left file (‘<‘) has this extra line

- **md5sum/sha*sum**: Hash files
  - Example use: Hash all static files, compare hashes later (e.g. using diff)

```
$ find /path -exec sha256sum '{}' ';' > SHA256SUM.orig
    ... (time passes) ...
$ find /path -exec sha256sum '{}' ';' > SHA256SUM.now
$ diff SHA256SUM.orig SHA256SUM.now
```

- **dd**: Do block IO with fine degree of control
  - Example use: Overwrite the first 1MB of a hard drive (destroys filesystem, but data is still intact – insecure but fast drive erasure)

```
$ dd if=/dev/zero of=/dev/sda bs=1k count=1k
```

More common commands (2)

- **hd**/**hexdump**/**od**: Hex dump (comes in a few variants)
  - Example use: Examine a config file for non-printable or Unicode characters that may be triggering a parser bug.

```
$ hd config1
00000000  73 65 74 74 69 6e 67 31  3d 79 65 73 ff 0a 73 65  |setting1=yes..se|
00000010  74 74 69 6e 67 32 3d 6f  6b 0a                    |tting2=ok.|
0000001a
```

- **strings**: Scan an otherwise binary file for printable strings
  - Example use: Quickly assess an unknown binary file for clues as to its nature

```
$ strings setup.exe | less
```

(scroll through lots of content quickly)

```
<assemblyIdentity version="1.0.0.0" processorArchitecture="X86"
name="DS.SolidWorks.setup" type="win32"></assemblyIdentity><description>This file
will allow SolidWorks to take advantage of xp themes.</description>
```

Conclusion: this is an installer for SolidWorks.

More common commands (3)

- `file`: Identify what kind of file you have by its format
  - Example use: Attacker pulled down an opaque file, what is it?

```
$ file hax.dat
dat: gzip compressed data, last modified: Thu Aug  9 16:50:37 2018, from Unix
$ gzip -cd hax.dat | file -
/dev/stdin: PE32+ executable (console) x86-64, for MS Windows
```

Most programs that take a filename can take '-' to mean stdin.

Conclusion: It's a gzip'd Windows executable

- `wget`/`curl`: Fetch internet stuff via HTTP (and other protocols)
  - **wget** downloads to file by default, **curl** writes to stdout by default (but either can do the other with options)
  - Example use 1: Download a file

```
$ wget http://150.2.3.5/attacker-kit.tgz
```

  - Example use 2: Hit a web API (the URL below tells you your external IP)

```
$ curl http://dsss.be/ip/
152.3.64.179
vcm-292.vm.duke.edu
```

Examples (1)

- Quick SSH banner recon:

It's like echo, but it's printf.

```
$ for H in `cat hostlist` ; do printf "%-30s" "$H" ; echo hi | nc $H 22 | head
-n1 ; done
remote.eos.ncsu.edu           SSH-2.0-OpenSSH_7.4
x.dsss.be                     SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
dsss.be                       SSH-2.0-OpenSSH_6.6.1p1 Ubuntu-2ubuntu2.10
reliant.colab.duke.edu        SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
davros.egr.duke.edu           SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
esa00.egr.duke.edu            SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
esa01.egr.duke.edu            SSH-2.0-OpenSSH_7.6p1 Ubuntu-4
storemaster.egr.duke.edu      SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.4
```

Examples (2)

- Download all the course notes (well, all linked PDFs):

```
$ wget -r -l1 -A pdf http://people.duke.edu/~tkb13/courses/ece590-sec/
$ find
.
./people.duke.edu
./people.duke.edu/~tkb13
./people.duke.edu/~tkb13/courses
./people.duke.edu/~tkb13/courses/ece590-sec
./people.duke.edu/~tkb13/courses/ece590-sec/slides
./people.duke.edu/~tkb13/courses/ece590-sec/slides/01-intro.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/slides/02-overview.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/slides/03-networking.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/slides/04-crypto.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/resources
./people.duke.edu/~tkb13/courses/ece590-sec/resources/appx
./people.duke.edu/~tkb13/courses/ece590-sec/resources/appx/C-Standards.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/resources/appx/F-TCP-IP.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/resources/appx/I-DomainNameSystem.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/homework
./people.duke.edu/~tkb13/courses/ece590-sec/homework/homework0.pdf
./people.duke.edu/~tkb13/courses/ece590-sec/homework/Ethics Pledge.pdf
./people.duke.edu/~tkb13/courses/ncsu-csc405-2015fa
```

Default behavior prints everything below here in the directory tree – a quick way to check what we got.

Examples (3)

Search a big directory tree for a file in old dBase format

- Using find’s -exec option:

```
$ find -exec file '{}' ';' | grep -i dbase
./server01-back/dat/cust20150501/dbase_03.dbf: FoxBase+/dBase III DBF, 14 records * 590,
update-date 05-7-13, at offset 1025 1st record "0507121     CMP   circular     12‟
```

-exec will run a command for each file found, with {} as the filename, terminating the command with ‘;’.

- Using xargs for efficiency (run fewer discrete processes):

```
$ find | xargs file | grep -i dbase
./server01-back/dat/cust20150501/dbase_03.dbf: FoxBase+/dBase III DBF, 14 records * 590,
update-date 05-7-13, at offset 1025 1st record "0507121     CMP   circular     12‟
```

xargs takes files in stdin and runs the given command on many of them at a time

- Using xargs with null delimiters to deal with filenames with spaces:

```
$ find -print0 | xargs -0 file | grep -i dbase
./server01-back/dat/cust20150501/spacey filename.dbf: FoxBase+/dBase III DBF, 14 records *
590, update-date 05-7-13, at offset 1025 1st record "0507121     CMP   circular     12‟
```

Both find’s output and xargs’s input are set to null-delimited instead of whitespace delimited.

Advanced uses of SSH

Advanced SSH: Tunnels

- Secure Shell (SSH): We know it logs you into stuff and is encrypted. *It does WAY MORE.*
- **SSH tunnels**: Direct TCP traffic through the SSH connection
  - `ssh -L <bindport>:<farhost>:<farport>  <server>`
    - **Local forward**: Opens port `bindport` on the local machine; any connection to it will *tunnel* through the SSH connection and cause `server` to connect to `farhost` on `farport`.
  - `ssh -R <bindport>:<nearhost>:<nearport>  <server>`
    - **Remote forward**: Opens port `bindport` on `server`; any connection to it will *tunnel* back through the SSH connection and the local machine will connect to `nearhost` on `nearport`.
  - `ssh -D <bindport>  <server>`
    - **Dynamic proxy**: Opens port `bindport` on the local machine. This port acts as a SOCKS proxy (a protocol allowing clients to open TCP connections to arbitrary hosts/ports); the proxy will exit on the server side. Browsers and other apps support SOCKS proxy protocol.
    - Easy way to punch into or out of a restricted network environment.

Advanced SSH: Tunnel examples

- Example **local forward**:
  - You want to connect to an app over the network, but it doesn't support encryption and/or you don't trust its security.
  - Solution:
    - Set app daemon to only listen on loopback connections (127.0.0.1) port 8888
    - SSH to server with local forward enabled:

      ```
      ssh -L 8888:localhost:8888 myserver.com
      ```
    - Connect your client to localhost:8888 instead of myserver.com:8888.
      All traffic is tunneled through encryption; access requires SSH creds.
- Example **remote forward**:
  - You're an attacker with SSH credentials to a machine behind a NAT. You have an exploit that lets you run a command on another machine behind the NAT.
  - Solution: SSH to a server you control with a reverse SSH forwarder:

    ```
    ssh -R 2222:victim:22 hackerserver.com
    ```
    - Can then connect to hackerserver.com's loopback port 2222 to get to victim.
- Example **dynamic proxy**: Turn it on. Set browser to use it. Surf via server.
  - Bypass censorship, do web-admin on a restricted network, tunnel through a NAT, etc.

Advanced SSH: Keys

- You’re used to using passwords to login. That’s...decent.

- **Alternative: SSH supports public/private key pairs!**
    - Pro: Allows passwordless login (or you can protect the key with a passphrase)
    - Pro: Key file is random and way longer than password (kills dictionary attack)
    - Pro: Can distribute your public key to any server you want easy access to
    - Con: Private key *must be kept secure!* It allows login!!

Advanced SSH: Key generation

- Create key pair:

`$ ssh-keygen` (can provide various options, but default are ok)

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tkbletsc/.ssh/id_rsa): mykey
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in mykey.
Your public key has been saved in mykey.pub.
The key fingerprint is:
SHA256:kywUn3nyI+LHOnsOYND5+FY7qIaTS+Ta0bXVjGTVY3Y tkbletsc@FREEMAN
The key's randomart image is:
+---[RSA 2048]----+
|      .   ..     |
|   . . o +  = E  |
|  . o . B .o o   |
|   . + + O       |
|  . + = S =      |
| o o = O + .     |
|  +o. B =        |
| ++..o.+..       |
|. o+. o=.        |
+----[SHA256]-----+
```

Advanced SSH: Key files

- Examining the keys:

```
$ cat mykey
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAq6vZKqVSLfZoiXd6yEgu3ZdLO/gv8mBaepWvJbISe5YKQw63
dBqnLAZc0rJcoqzHgwBjddWUyzDh7g7+MZYgf+n+xE+3QDchqdrktPxj96TMfWUZ
tH1tpY1UNdbIStAhMbGr/L6aKFs/Ouk5RhWw+GPA7N1diATD0SYibTqdG5+JQqGn

 ...

/4zTb3GDiXFIY9+raaFZ1XLJKBzfhi3ED4ga3nqmeKK60CDTvx8QbA==
-----END RSA PRIVATE KEY-----


$ cat mykey.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCrq9kqpVIt9miJd3rISC7dl0s7+C/yYFp6la8l
shJ7lgpDDrd0GqcsBlzSslyirMeDAGN11ZTLMOHuDv4xliB/6fuJK0D4BCFbhD8Y2eGh
TZ/l/g9uIwIv7merL+UQduCSKvqLo1X4JYsI5VSkNKCjcLo7lJoCOUazqmttkX2EBSGd
3VYp97Eu3XC3rqDAa/FnUe3E4w8nHLk9mB6/qbyr tkbletsc@FREEMAN
```

Informational comment, defaults to username@hostname, could be anything.

Advanced SSH: Key usage

- Authorizing a key:
  - Copy only **`mykey.pub`** to the remote machine you want to establish access to
  - On remote machine, add it to **`~/.ssh/authorized_keys`**:

```
$ cat mykey.pub >> ~/.ssh/authorized_keys
```

- Using a key to login:
  - Provite the *identity file* (private key) with **`-i`**:

```
$ ssh -i mykey remotehost.com
```

  - SSH will use **`~/.ssh/id_rsa`** by default – can use this “default key” without extra options.

- Remember: **keep your private key safe!!!**

Advanced SSH: Commands

- Can give a command with ssh to only do that command (no interactive session). Stdin/stdout/stderr are tunneled appropriately!
  - Really works great with passwordless keys!
  - Find out uptime of server quickly:

    ```
    $ ssh myserver uptime
    ```

  - Reboot nodes in a cluster:

    ```
    $ for A in node{0..7} ; do ssh root@$A reboot ; done
    ```

  - Back up remote physical disk image:

    ```
    $ ssh root@server bash -c "gzip -c < /dev/sda" > server.img.gz
    ```

Advanced SSH: SCP, SFTP, and Rsync

- Almost any SSH server is also a file server using the SFTP protocol. The `scp` command is one way to use this.
  - Copy a file to remote home directory:

    ```
    $ scp file1.txt username@myserver:
    ```

  - Copy a directory to remote server's web root:

    ```
    $ scp -r dir1/ webadmin@myserver:/var/www/
    ```

- Can also use a tool called `rsync` to copy *only changes* to files
  - Here's the script I use to update the course site:

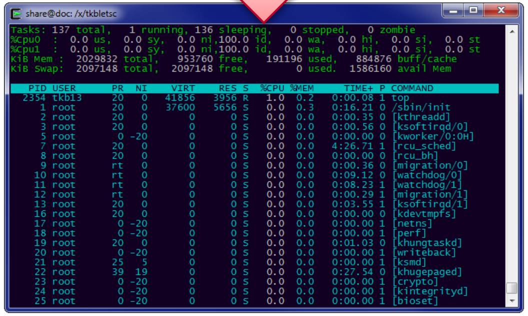
    ```
    echo COLLECTING COURSE SITES
    rsync -a --delete-delay ./ECE590-security/website/
      ./www/courses/ece590-sec/
    rsync -a --delete-delay ./ECE590-storage/website/
      ./www/courses/ece590-stor/


    rsync -va --progress --delete-delay --no-perms www/*
      tkb13@login.oit.duke.edu:public_html/
    ```

Understanding and controlling the terminal

Brief terminal history

- Original terminal: **the teletype machine**
  - Based on typewriter technology
  - It's why we say "carriage return" and "line feed"
- Then came: **the serial terminal**
  - CRT display with basic logic to speak serial protocol
  - Many hooked up to one mainframe
  - Needed new *codes* to do new things like "clear screen" and "underline" without breaking compatibility
- Now we have: **terminal emulators** like **`xterm`**
  - Even more *codes* to do color, cursor movement, non-scrolling regions, etc.

- In Linux, the physical display is a "TTY" (teletype), e.g. `/dev/tty1`.
- Logical terminals like SSH sessions are "pseudoterminals", e.g. `/dev/pts/0`

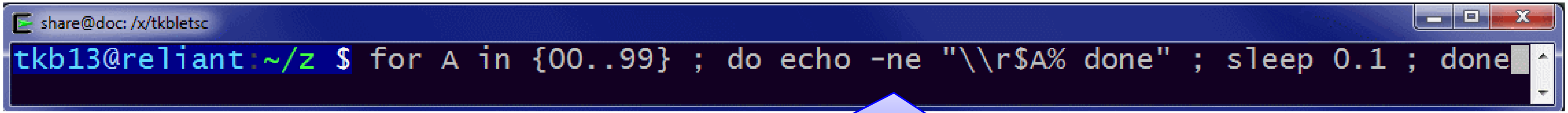


```
Tasks: 137 total,   1 running, 136 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2029832 total,   953760 free,   191196 used,   884876 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used.  1586160 avail Mem

  PID USER      PR  NI    VIRT    RES S  %CPU %MEM     TIME+ P COMMAND
 2354 tkb13     20   0   41856   3956 R   1.0  0.2   0:00.08 1 top
    1 root      20   0   37600   5656 S   0.0  0.3   0:16.21 0 /sbin/init
    2 root      20   0       0      0 S   0.0  0.0   0:00.35 0 [kthreadd]
    3 root      20   0       0      0 S   0.0  0.0   0:00.56 0 [ksoftirqd/0]
    5 root       0 -20       0      0 S   0.0  0.0   0:00.00 0 [kworker/0:0H]
    7 root      20   0       0      0 S   0.0  0.0   4:26.71 1 [rcu_sched]
    8 root      20   0       0      0 S   0.0  0.0   0:00.00 0 [rcu_bh]
    9 root      rt   0       0      0 S   0.0  0.0   0:00.36 0 [migration/0]
   10 root      rt   0       0      0 S   0.0  0.0   0:09.12 0 [watchdog/0]
   11 root      rt   0       0      0 S   0.0  0.0   0:08.23 1 [watchdog/1]
   12 root      rt   0       0      0 S   0.0  0.0   0:00.29 1 [migration/1]
   13 root      20   0       0      0 S   0.0  0.0   0:03.55 1 [ksoftirqd/1]
   16 root      20   0       0      0 S   0.0  0.0   0:00.00 0 [kdevtmpfs]
   17 root       0 -20       0      0 S   0.0  0.0   0:00.00 1 [netns]
   18 root       0 -20       0      0 S   0.0  0.0   0:00.00 1 [perf]
   19 root      20   0       0      0 S   0.0  0.0   0:01.03 0 [khungtaskd]
   20 root       0 -20       0      0 S   0.0  0.0   0:00.00 1 [writeback]
   21 root      25   5       0      0 S   0.0  0.0   0:00.00 1 [ksmd]
   22 root      39  19       0      0 S   0.0  0.0   0:27.54 0 [khugepaged]
   23 root       0 -20       0      0 S   0.0  0.0   0:00.00 1 [crypto]
   24 root       0 -20       0      0 S   0.0  0.0   0:00.00 1 [kintegrityd]
   25 root       0 -20       0      0 S   0.0  0.0   0:00.00 1 [bioset]
```

Terminal control sequences: Basic idea

- Some ASCII values are special:
  - 0x0A = Linefeed (move cursor to next line), written as **`\n`**
  - 0x0D = Carriage return (move cursor to left), written as **`\r`**
  - 0x07 = Bell (beep), written as **`\a`**
  - 0x1B = Escape – indicates a special multi-byte sequence, written as **`\e`**
    - MANY sequences exist. Full documentation here.

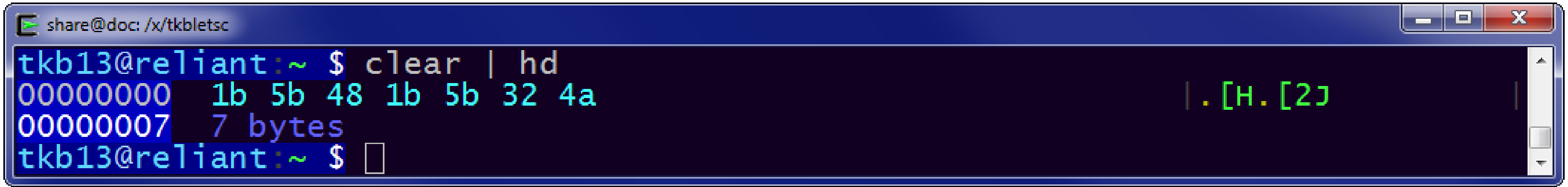
- Example: Show a progress line without scrolling:



```
tkb13@reliant:~/z $ for A in {00..99} ; do echo -ne "\\r$A% done" ; sleep 0.1 ; done
```

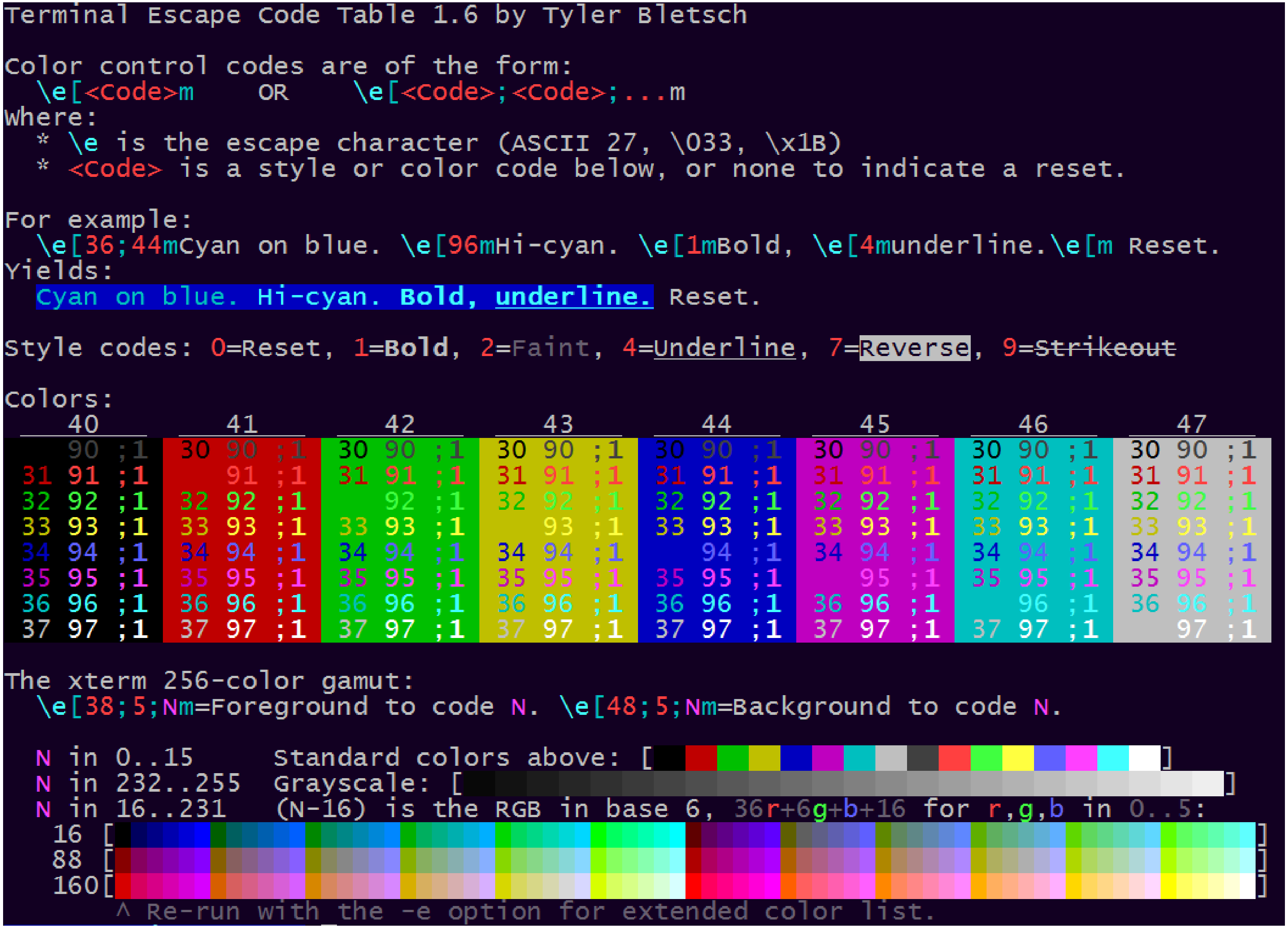
-n means "no newline", -e means "allow escape characters".

- Example: How does the 'clear' command work?



```
tkb13@reliant:~ $ clear | hd
00000000  1b 5b 48 1b 5b 32 4a                              |.[H.[2J|
00000007  7 bytes
tkb13@reliant:~ $ 
```

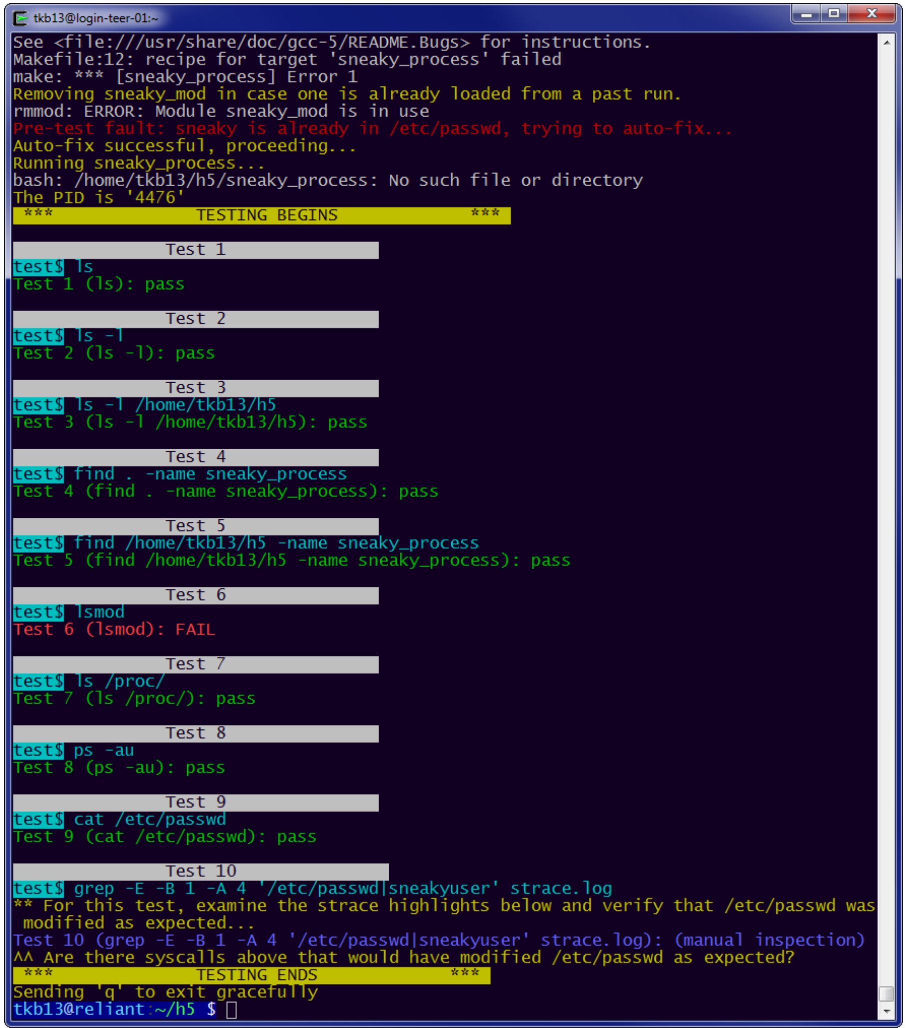
Terminal control sequences: Color!



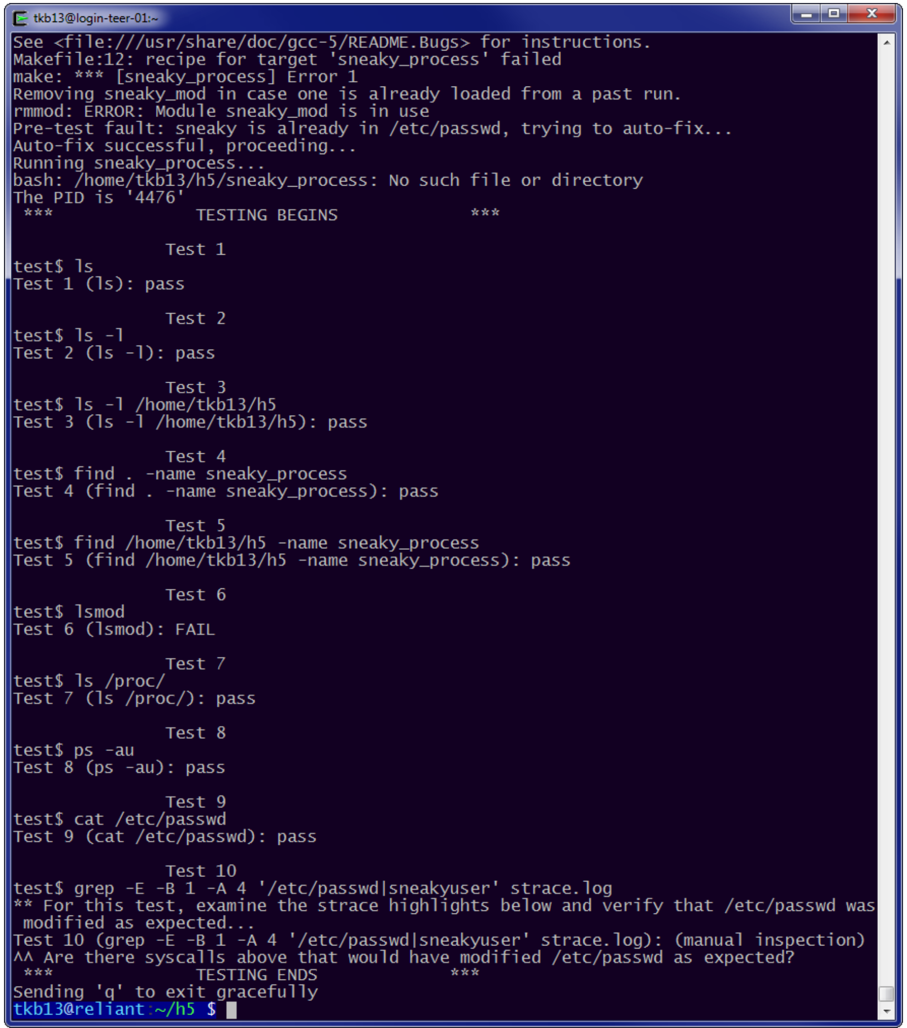
```
Terminal Escape Code Table 1.6 by Tyler Bletsch

Color control codes are of the form:
  \e[<Code>m    OR    \e[<Code>;<Code>;...m
Where:
  * \e is the escape character (ASCII 27, \033, \x1B)
  * <Code> is a style or color code below, or none to indicate a reset.

For example:
  \e[36;44mCyan on blue. \e[96mHi-cyan. \e[1mBold, \e[4munderline.\e[m Reset.
Yields:
  Cyan on blue. Hi-cyan. Bold, underline. Reset.

Style codes: 0=Reset, 1=Bold, 2=Faint, 4=Underline, 7=Reverse, 9=~~Strikeout~~

Colors:
     40        41        42        43        44        45        46        47
 ________  ________  ________  ________  ________  ________  ________  ________
    90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1  30 90 ;1
 31 91 ;1     91 ;1  31 91 ;1  31 91 ;1  31 91 ;1  31 91 ;1  31 91 ;1  31 91 ;1
 32 92 ;1  32 92 ;1     92 ;1  32 92 ;1  32 92 ;1  32 92 ;1  32 92 ;1  32 92 ;1
 33 93 ;1  33 93 ;1  33 93 ;1     93 ;1  33 93 ;1  33 93 ;1  33 93 ;1  33 93 ;1
 34 94 ;1  34 94 ;1  34 94 ;1  34 94 ;1     94 ;1  34 94 ;1  34 94 ;1  34 94 ;1
 35 95 ;1  35 95 ;1  35 95 ;1  35 95 ;1  35 95 ;1     95 ;1  35 95 ;1  35 95 ;1
 36 96 ;1  36 96 ;1  36 96 ;1  36 96 ;1  36 96 ;1  36 96 ;1     96 ;1  36 96 ;1
 37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1  37 97 ;1     97 ;1

The xterm 256-color gamut:
  \e[38;5;Nm=Foreground to code N. \e[48;5;Nm=Background to code N.

  N in 0..15      Standard colors above: [████████████████]
  N in 232..255   Grayscale: [████████████████████████]
  N in 16..231    (N-16) is the RGB in base 6, 36r+6g+b+16 for r,g,b in 0..5:
   16 [████████████████████████████████████████████████████████████████████████]
   88 [████████████████████████████████████████████████████████████████████████]
   160[████████████████████████████████████████████████████████████████████████]
      ^ Re-run with the -e option for extended color list.
```

Why bother?

- Making output visually distinctive can *greatly* accelerate a task!
- Tester for ECE650 root kit: which would you rather use?



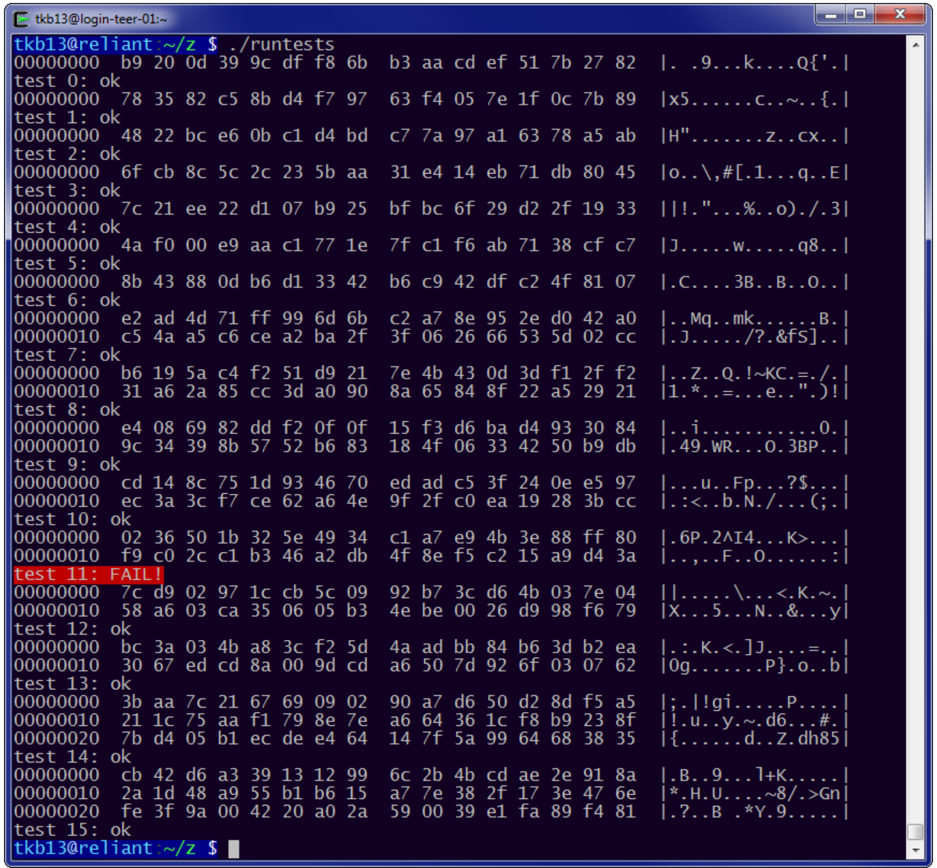
```
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
Makefile:12: recipe for target 'sneaky_process' failed
make: *** [sneaky_process] Error 1
Removing sneaky_mod in case one is already loaded from a past run.
rmmod: ERROR: Module sneaky_mod is in use
Pre-test fault: sneaky is already in /etc/passwd, trying to auto-fix...
Auto-fix successful, proceeding...
Running sneaky_process...
bash: /home/tkb13/h5/sneaky_process: No such file or directory
The PID is '4476'
 ***                 TESTING BEGINS              ***

                Test 1
test$ ls
Test 1 (ls): pass

                Test 2
test$ ls -l
Test 2 (ls -l): pass

                Test 3
test$ ls -l /home/tkb13/h5
Test 3 (ls -l /home/tkb13/h5): pass

                Test 4
test$ find . -name sneaky_process
Test 4 (find . -name sneaky_process): pass

                Test 5
test$ find /home/tkb13/h5 -name sneaky_process
Test 5 (find /home/tkb13/h5 -name sneaky_process): pass

                Test 6
test$ lsmod
Test 6 (lsmod): FAIL

                Test 7
test$ ls /proc/
Test 7 (ls /proc/): pass

                Test 8
test$ ps -au
Test 8 (ps -au): pass

                Test 9
test$ cat /etc/passwd
Test 9 (cat /etc/passwd): pass

                Test 10
test$ grep -E -B 1 -A 4 '/etc/passwd|sneakyuser' strace.log
** For this test, examine the strace highlights below and verify that /etc/passwd was
 modified as expected...
Test 10 (grep -E -B 1 -A 4 '/etc/passwd|sneakyuser' strace.log): (manual inspection)
^^ Are there syscalls above that would have modified /etc/passwd as expected?
 ***                 TESTING ENDS             ***
Sending 'q' to exit gracefully
tkb13@reliant:~/h5 $
```



```
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
Makefile:12: recipe for target 'sneaky_process' failed
make: *** [sneaky_process] Error 1
Removing sneaky_mod in case one is already loaded from a past run.
rmmod: ERROR: Module sneaky_mod is in use
Pre-test fault: sneaky is already in /etc/passwd, trying to auto-fix...
Auto-fix successful, proceeding...
Running sneaky_process...
bash: /home/tkb13/h5/sneaky_process: No such file or directory
The PID is '4476'
 ***                 TESTING BEGINS              ***

               Test 1
test$ ls
Test 1 (ls): pass

               Test 2
test$ ls -l
Test 2 (ls -l): pass

               Test 3
test$ ls -l /home/tkb13/h5
Test 3 (ls -l /home/tkb13/h5): pass

               Test 4
test$ find . -name sneaky_process
Test 4 (find . -name sneaky_process): pass

               Test 5
test$ find /home/tkb13/h5 -name sneaky_process
Test 5 (find /home/tkb13/h5 -name sneaky_process): pass

               Test 6
test$ lsmod
Test 6 (lsmod): FAIL

               Test 7
test$ ls /proc/
Test 7 (ls /proc/): pass

               Test 8
test$ ps -au
Test 8 (ps -au): pass

               Test 9
test$ cat /etc/passwd
Test 9 (cat /etc/passwd): pass

               Test 10
test$ grep -E -B 1 -A 4 '/etc/passwd|sneakyuser' strace.log
** For this test, examine the strace highlights below and verify that /etc/passwd was
 modified as expected...
Test 10 (grep -E -B 1 -A 4 '/etc/passwd|sneakyuser' strace.log): (manual inspection)
^^ Are there syscalls above that would have modified /etc/passwd as expected?
 ***                 TESTING ENDS             ***
Sending 'q' to exit gracefully
tkb13@reliant:~/h5 $
```

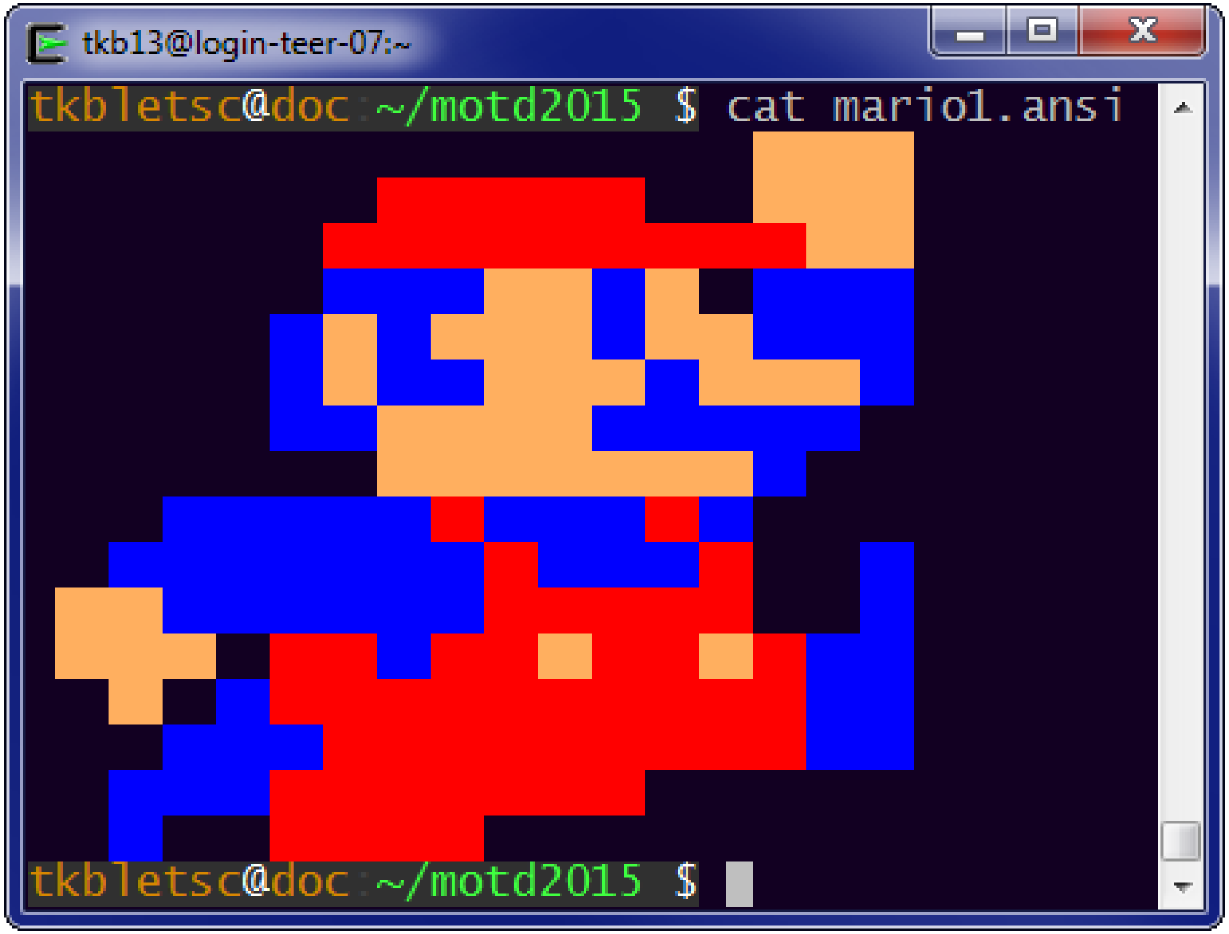
Simple example – make errors obvious

```
for testnum in {0..15} ; do
  if ./dotest $testnum ; then
    echo "test $testnum: ok"
  else
    echo -e "\e[41mtest $testnum: FAIL!\e[m"
  fi
done
```



```
tkb13@reliant:~/z $ ./runtests
00000000  b9 20 0d 39 9c df f8 6b  b3 aa cd ef 51 7b 27 82  |. .9...k....Q{'.|
test 0: ok
00000000  78 35 82 c5 8b d4 f7 97  63 f4 05 7e 1f 0c 7b 89  |x5......c..~..{.|
test 1: ok
00000000  48 22 bc e6 0b c1 d4 bd  c7 7a 97 a1 63 78 a5 ab  |H"......z..cx..|
test 2: ok
00000000  6f cb 8c 5c 2c 23 5b aa  31 e4 14 eb 71 db 80 45  |o..\,#[.1...q..E|
test 3: ok
00000000  7c 21 ee 22 d1 07 b9 25  bf bc 6f 29 d2 2f 19 33  ||!."...%..o)./.3|
test 4: ok
00000000  4a f0 00 e9 aa c1 77 1e  7f c1 f6 ab 71 38 cf c7  |J.....w.....q8..|
test 5: ok
00000000  8b 43 88 0d b6 d1 33 42  b6 c9 42 df c2 4f 81 07  |.C....3B..B..O..|
test 6: ok
00000000  e2 ad 4d 71 ff 99 6d 6b  c2 a7 8e 95 2e d0 42 a0  |..Mq..mk......B.|
00000010  c5 4a a5 c6 ce a2 ba 2f  3f 06 26 66 53 5d 02 cc  |.J...../?.&fS]..|
test 7: ok
00000000  b6 19 5a c4 f2 51 d9 21  7e 4b 43 0d 3d f1 2f f2  |..Z..Q.!~KC.=./.|
00000010  31 a6 2a 85 cc 3d a0 90  8a 65 84 8f 22 a5 29 21  |1.*..=...e..".)!|
test 8: ok
00000000  e4 08 69 82 dd f2 0f 0f  15 f3 d6 ba d4 93 30 84  |..i...........0.|
00000010  9c 34 39 8b 57 52 b6 83  18 4f 06 33 42 50 b9 db  |.49.WR...O.3BP..|
test 9: ok
00000000  cd 14 8c 75 1d 93 46 70  ed ad c5 3f 24 0e e5 97  |...u..Fp...?$...|
00000010  ec 3a 3c f7 ce 62 a6 4e  9f 2f c0 ea 19 28 3b cc  |.:<..b.N./...(;.|
test 10: ok
00000000  02 36 50 1b 32 5e 49 34  c1 a7 e9 4b 3e 88 ff 80  |.6P.2^I4...K>...|
00000010  f9 c0 2c c1 b3 46 a2 db  4f 8e f5 c2 15 a9 d4 3a  |..,..F..O......:|
test 11: FAIL!
00000000  7c d9 02 97 1c cb 5c 09  92 b7 3c d6 4b 03 7e 04  ||.....\...<.K.~.|
00000010  58 a6 03 ca 35 06 05 b3  4e be 00 26 d9 98 f6 79  |X...5...N..&...y|
test 12: ok
00000000  bc 3a 03 4b a8 3c f2 5d  4a ad bb 84 b6 3d b2 ea  |.:.K.<.]J....=..|
00000010  30 67 ed cd 8a 00 9d cd  a6 50 7d 92 6f 03 07 62  |0g.......P}.o..b|
test 13: ok
00000000  3b aa 7c 21 67 69 09 02  90 a7 d6 50 d2 8d f5 a5  |;.|!gi.....P....|
00000010  21 1c 75 aa f1 79 8e 7e  a6 64 36 1c f8 b9 23 8f  |!.u..y.~.d6...#.|
00000020  7b d4 05 b1 ec de e4 64  14 7f 5a 99 64 68 38 35  |{......d..Z.dh85|
test 14: ok
00000000  cb 42 d6 a3 39 13 12 99  6c 2b 4b cd ae 2e 91 8a  |.B..9...l+K.....|
00000010  2a 1d 48 a9 55 b1 b6 15  a7 7e 38 2f 17 3e 47 6e  |*.H.U....~8/.>Gn|
00000020  fe 3f 9a 00 42 20 a0 2a  59 00 39 e1 fa 89 f4 81  |.?..B .*Y.9.....|
test 15: ok
tkb13@reliant:~/z $
```

Also you can do cool crap



```
tkbletsc@doc ~/motd2015 $ cat mario1.ansi
tkbletsc@doc ~/motd2015 $ 
```

Scripting languages and regular expressions

Regular expression material is adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan and “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Higher-level scripting languages

- Key languages categories commonly used:
  - **Application**: Java, C#, maybe C++
  - **Systems programming**: C, maybe Rust
  - **Shell**: bash (or ksh, tcsh, etc.)
  - **Scripting**: Python, Perl, or Ruby
- You *can* do everything in bash, but it gets ugly. Things bash is awkward at:
  - Math
  - Arrays
  - Hash/dictionary data structures
  - Really any data structures...
- Turn to **scripting languages**: dynamic, interpreted, compact

Scripting language key insight: three fundamental types

- Most data manipulation tasks can be phrased as simple algorithms against these three types:
  - **Scalar**: simple value, numeric or string
  - **Array**: list of values (can nest)
  - **Hash/dictionary/map**: relationship between keys and values (can nest)

Perl

```
my %pairs = ( "hello" => 13,
              "world" => 31,
              "!" => 71 );
foreach my $key ( keys %pairs ) {
    print "key = $key, value = $pairs{$key}\n";
}
```

python

```
myDict = { "hello": 13,
           "world": 31,
           "!"    : 71 }
for key, value in myDict.items():
    print ("key = %s, value = %s" % (key, value))
```

Examples from here.

Ruby

```
my_dict = { "hello" => 13,
            "world" => 31,
            "!"     => 71 }
my_dict.each {|key, value| puts "key = #{key}, value = #{value}"}
```

One-liners

- Scripting languages support one-liners (typed from shell as a single command).
- Perl is the king of one-liners.
  - **`-e`** to provide code
  - **`-n`** automatically wraps code in “for each line of input from stdin or files”
  - **`-i`** replaces the content of given files with stdout of program (can provide filename extension to back up original data to)
- Quick Perl intro
  - Scalar variables start with a dollar sign, e.g. **`$var`**
  - Most functions, if you don’t specify, affect a variable called **`$_`**
  - Reference an array element value with **`$array[$i]`**, whole array is **`@array`**
  - Reference a hash element value with **`$hash{$k}`**, whole hash is **`%hash`**
  - Variables you make reference to are automatically created if they don’t exist (including arrays and hashes)
  - One-line comments with **#**

Perl one-liner example

- Remove duplicate lines from a file while preserving original order

Long-winded Perl:

```
  while (<>) {   # for each line
    if (!$hash{$_}) {
      print;
    }
    $hash{$_}=1;
  }
```

Run it:

```
$ perl dedupe.pl in.txt
```

One-liner:

```
$ perl -ne 'if (!$h{$_}){print} $h{$_}=1;' in.txt
alpha
delta
bravo
charlie
```

Crazy dense one-liner:

```
$ perl -ne '$h{$_}++||print;' in.txt
```

in.txt

```
alpha
delta
alpha
bravo
bravo
alpha
charlie
alpha
bravo
alpha
```

Manipulating text

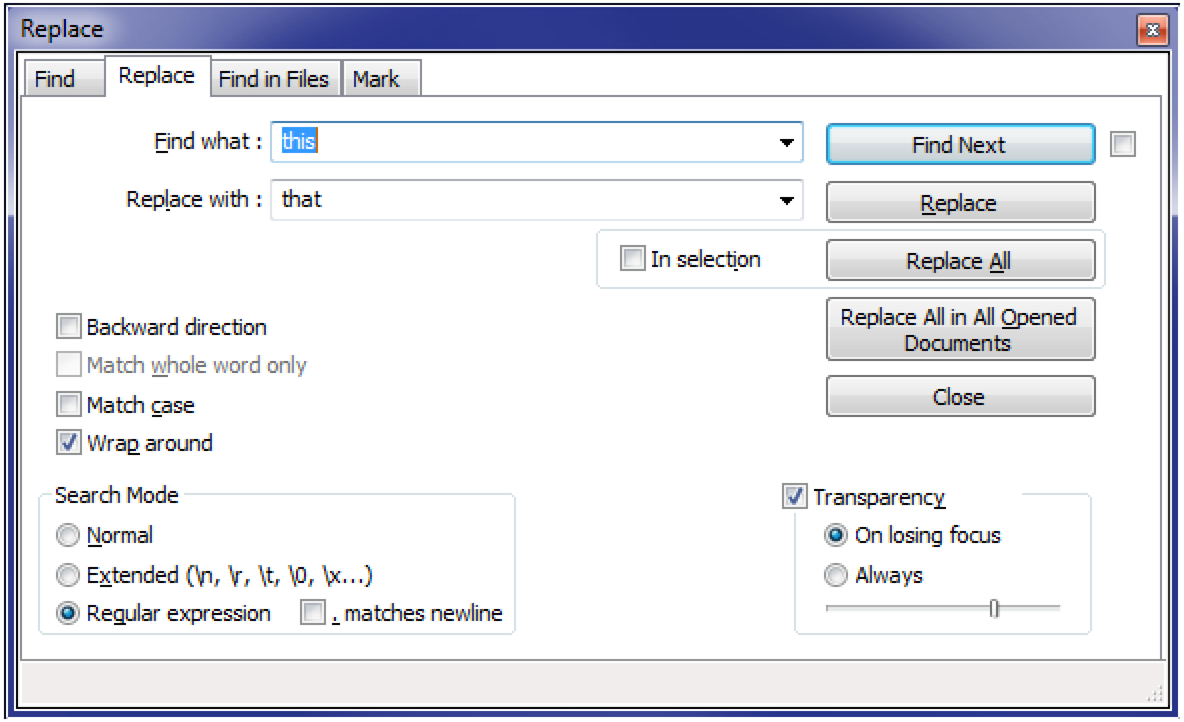
- Task: extract the hostname part of a URL, e.g. http://google.com/images
- Thought process:
  - Idea: Start at character 7, capture until you find a slash
  - Problem: what about https?
  - Idea: Go until you see two slashes in a row, then capture until you find a slash
  - Problem: can have more than two slashes at start
  - Idea: Go until you see two or more slashes in a row, then capture until you find a slash
  - Problem: What about username specifier (user@) and port number (:80)?
  - ugh nevermind just give up ☹
- Solution: We need a **language** to describe string processing!

THE LANGUAGE OF STRING PROCESSING

REGULAR EXPRESSIONS

Regular Expressions

- Regular expressions are expressive rules for walking a string
  - May capture parts of the string (parsing) or modify it (substitution)
  - Like a fancy find-and-replace



Find what : this

Replace with : that

□ In selection

□ Backward direction

□ Match whole word only

□ Match case

☑ Wrap around

Search Mode

○ Normal

○ Extended (\n, \r, \t, \0, \x...)

● Regular expression □ . matches newline

☑ Transparency

● On losing focus

○ Always

Understanding Regular Expressions

- Very powerful, cryptic, and fun
- Regular expressions are a language:
  - Based on "marker characters" - programming with characters

- The "gold standard" variant is from Perl:
  Perl-Compatible Regular Expressions (PCRE)
- *Many* languages support Perl-Compatible Regular Expressions:
  Perl, **grep** (with -P), **sed** (mostly), **Python**, **Ruby**, **Java**,
  most text editors, basically any language/tool worth using.

- **Common among languages**: Actual syntax *inside* regex
- **Differors between languages**: Syntax to call a regex, get feedback from it, provide options, etc.

- We'll use both Perl and Python examples – easy to port to others

Adapted from "Regular Expressions" in "Python for Informatics: Exploring Information" by Charles Severance at Univ. Michigan

Introduction to Regular Expressions

- Basic syntax
  - In Perl and sed, RegEx statements begin and end with `/`
    (This is language syntax, not the case for Python and others)
    - `/something/`
  - Escaping reserved characters is crucial
    - `/(i.e. /` is invalid because `(` must be closed
    - However, `/\(i\.e\. /` is valid for finding ‘(i.e. ’
    - Reserved characters include:

      `. * ? + ( ) [ ] { } / \ |`

      - Also some characters have special meanings based on their position in the statement

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Matching

- Text Matching
  - A RegEx can match plain text
    - ex. `if ($name =~ /Dan/) { print “match”; }`
    - But this will match Dan, Danny, McDaniel, etc…
- Full Text Matching with Anchors
  - Might want to match a whole line (or string)
    - ex. `if ($name =~ /^Dan$/) { print “match”; }`
    - This will only match Dan
    - ^ anchors to the front of the line
    - $ anchors to the end of the line

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

In Python: The Regular Expression Module

- Before you can use regular expressions in your program, you must import the library using "`import re`"
- Use `re.search()` to see if a string matches a regex
- Use `re.findall()` extract parts of a string that match your regex
- Use `re.sub()` to replace a regex match with another string
- Use `re.split()` to separate a string by a regex separator

- Example:
  - `if re.search(r'Dan', name): print "match"`

In Python, r-quotes mean “raw string”, i.e. “don’t interpret escapes in this string”, which makes it convenient to write Regexes which use all sorts of weird punctuation

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

General operation

- Engine searches string from the beginning
  - Plain text is treated literally
  - Special characters allow more flexible matching

- A regex is just a way to write a finite state machine (FSM)
  - FSM proceeds through states as matching characters are encountered; if a full regex is walked, that’s a match.

- Every character matters!
  - **`/  s/`** is not the same as **`/   s/`**

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Char Classes

- Allows specification of only certain allowable chars
  - `[dofZ]` matches only the letters d, o, f, and Z
    - If you have a string ‘dog’ then `/[dofZ]/` would match ‘d’ only even though ‘o’ is also in the class
    - So this expression can be stated “match one of either d, o, f, or Z.”
  - `[A-Za-z]` matches any letter
  - `[a-fA-F0-9]` matches any hexadecimal character
  - `[^*$/\\]` matches anything BUT *, $, /, or \
    - The ^ in the front of the char class specifies ‘not’
    - In a char class, you only need to escape: `\ ( ] - ^`

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Char Classes

- Special character classes match specific characters
  - `\d` matches a single digit
  - `\w` matches a word character: `[A-Za-z0-9_]`
  - `\b` matches a word boundary, e.g. `/\bword\b/`
  - `\s` matches a whitespace character (space, tab, newline)
  - `.` wildcard matches everything but newlines (can make it include newlines)
    - Use very carefully, you could get anything!

- To match "anything but…" capitalize the char class
  - i.e. `\D` matches anything that isn't a digit

Adapted from "Regular Expressions" by Ian Paterson at Rochester Institute of Technology

Regular Expression Char Classes

- Character Class Examples
  - `/e\w\w/`
    - Matches ear, eye, etc
  - `$thing = ‘1, 2, 3 strikes!’;  $thing =~ /\s\d/;`
    - Matches ‘ 2’
  - `$thing = ‘1, 2, 3 strikes!’;  $thing =~ /[\s\d]/;`
    - Matches ‘1’
  - Not always useful to match single characters
    - `$phone =~ /\d\d\d-\d\d\d-\d\d\d\d/;`
    - There’s got to be a better way…

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Repetition

- Repetition allows for flexibility
  - Range of occurrences
    - `$weight =~ /\d{2,3}/;`
      - Matches any weight from 10 to 999
    - `$name =~ /\w{5,}/;`
      - Matches any name longer than 5 letters
    - `if ($SSN =~ /\d{9}/) { print “Invalid SSN!”; }`
      - Matches exactly 9 digits

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Repetition

- General Quantifiers
  - Some more special characters
    - `$favoriteNumber =~ /\d*/;`
      - Matches any size number or no number at all
    - `$firstName =~ /\w+/;`
      - Matches one or more characters
    - `$middleInitial =~ /\w?/;`
      - Matches one or zero characters

Regular Expression Repetition

- Greedy vs Non-greedy matching
  - Greedy matching gets the longest results possible
  - Non-greedy matching gets the shortest possible
  - Let’s say **`$robot = ‘The12thRobotIs2ndInLine’`**
    - **`$robot =~ /\w*\d+/;`** (greedy)
      - Matches **`The12thRobotIs2`**
      - Maximizes the length of **`\w`**
    - **`$robot =~ /\w*?\d+/;`** (non-greedy)
      - *Add a ‘`?`’ to a repetition to make it non-greedy!*
      - Matches **`The12`**
      - Minimizes the length of **`\w`**

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Repetition

- Greedy vs Nongreedy matching
  - Suppose **`$txt = ‘something is so cool’`**
    - **`$txt =~ /something/;`**
      - Matches ‘something’
    - **`$txt =~ /so(mething)?/;`**
      - Matches ‘something’ and the second ‘so’
      - Parenthesis can be used for **grouping** (e.g. being modified by ‘?’) and **capture** (covered later)

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Regular Expression Real Life Examples

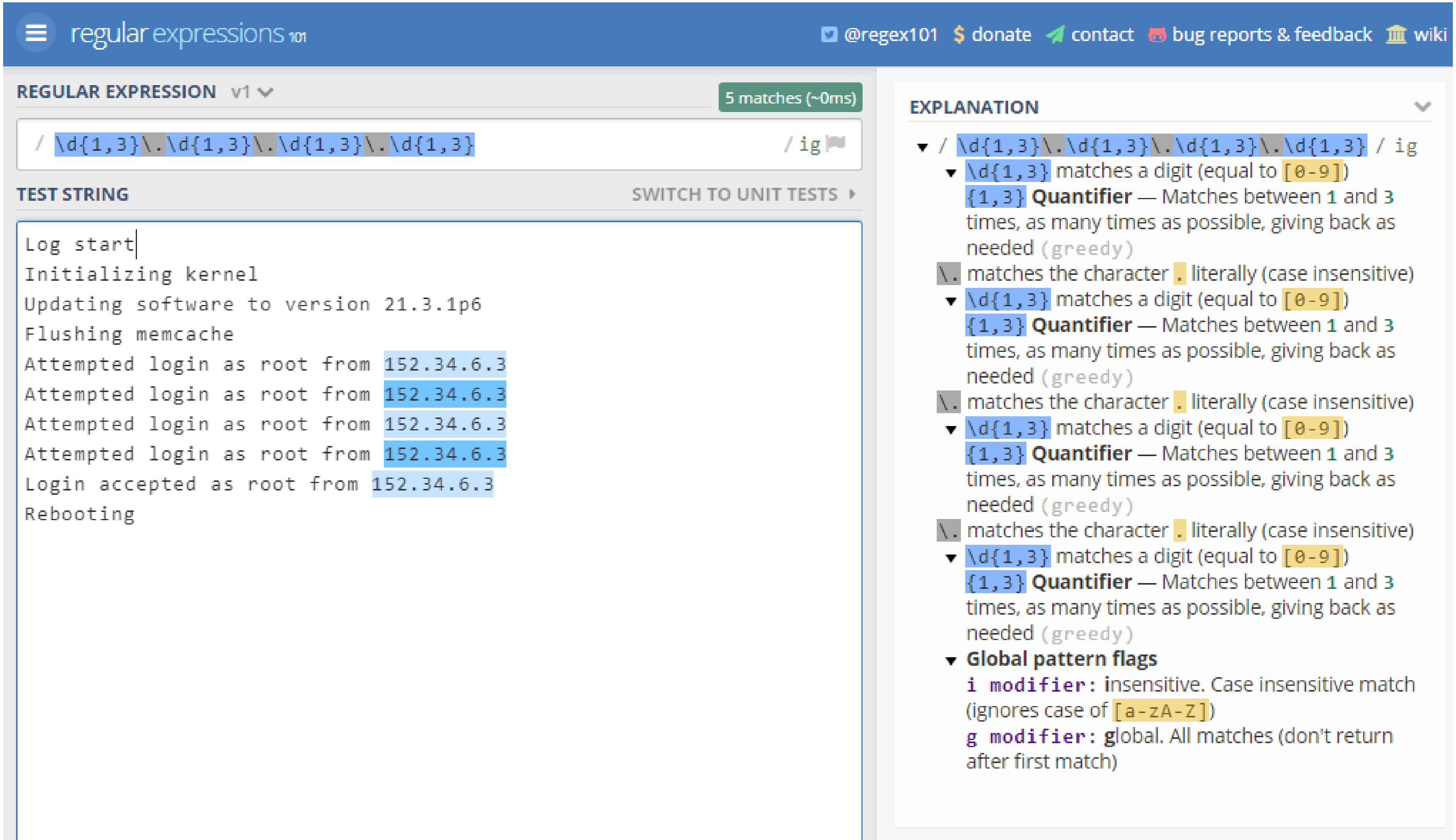
- Using what you've learned so far, you can…
- Validate an email address (note: regex below is a little oversimplified)
  - `$email =~ /^\w+@(\w+\.)*(\w+)$/`
- Determine if log entry includes an IPv4 address
  - `/\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}/`

- Regular expressions can be hard to write and even harder to read
- Two techniques can help:
  - Languages have various 'verbose' or 'extended' modes so that a regex can be multiple lines, include comments, etc.
  - Can use an interactive regex editor such as http://regex101.com/

Adapted from "Regular Expressions" by Ian Paterson at Rochester Institute of Technology

Regex101 example

- The IP address example



```
/ \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3} / ig
```

```
Log start
Initializing kernel
Updating software to version 21.3.1p6
Flushing memcache
Attempted login as root from 152.34.6.3
Attempted login as root from 152.34.6.3
Attempted login as root from 152.34.6.3
Attempted login as root from 152.34.6.3
Login accepted as root from 152.34.6.3
Rebooting
```

- / \d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3} / ig
  - \d{1,3} matches a digit (equal to [0-9])
    {1,3} **Quantifier** — Matches between 1 and 3 times, as many times as possible, giving back as needed (greedy)
  - \. matches the character . literally (case insensitive)
  - \d{1,3} matches a digit (equal to [0-9])
    {1,3} **Quantifier** — Matches between 1 and 3 times, as many times as possible, giving back as needed (greedy)
  - \. matches the character . literally (case insensitive)
  - \d{1,3} matches a digit (equal to [0-9])
    {1,3} **Quantifier** — Matches between 1 and 3 times, as many times as possible, giving back as needed (greedy)
  - \. matches the character . literally (case insensitive)
  - \d{1,3} matches a digit (equal to [0-9])
    {1,3} **Quantifier** — Matches between 1 and 3 times, as many times as possible, giving back as needed (greedy)
  - **Global pattern flags**
    i modifier: insensitive. Case insensitive match (ignores case of [a-zA-Z])
    g modifier: global. All matches (don't return after first match)

Alternation

- Alternation allows multiple possibilities
  - Let `$story = 'He went to get his mother'`

    ```
    $story =~ /^(He|She)\b.*?\b(his|her)\b.*? (mother|father|brother|sister|dog)/;
    ```

    - Also matches ‘She punched her fat brother’
  - Make sure the grouping is correct!

    ```
    $ans =~ /^(true|false)$/
    ```

    - Matches only ‘true’ or ‘false’

    `$ans =~ /^true|false$/` (same as `/(^true|false$)/`)

    - Matches ‘true never’ or ‘not really false’

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Grouping for Backreferences

- **Backreferences** (also known as **capture groups**)
  - We want to know what the expression finally ended up matching
    - Parenthesis give you **backreferences** let you see what was matched
    - Can be used *after* the expression has evaluated or even *inside* the expression itself!
    - Handled differently in different languages
    - Numbered from left to right, starting at 1

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Grouping for Backreferences

- Perl backreferences
  - Used inside the expression
    - `$txt =~ /\b(\w+)\s+\1\b/`
    - Finds any duplicated word, must use `\1` here (true in most languages)
  - Used after the expression
    - `$class =~ /(.+?)-(\d+)/`
    - The first word between hyphens is stored in the Perl variable `$1` (not `\1`) and the number goes in `$2`. (This part varies between languages)
    - `print “I am in class $1, section $2”;`
- Equivalent Python:

```
import re
cls = "ECE590-02"
m = re.match(r'(.+?)-(\d+)',cls)
print "I'm in class "+m.group(1)+", section "+m.group(2)
```

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Example: Email Headers

- Here are some email headers.

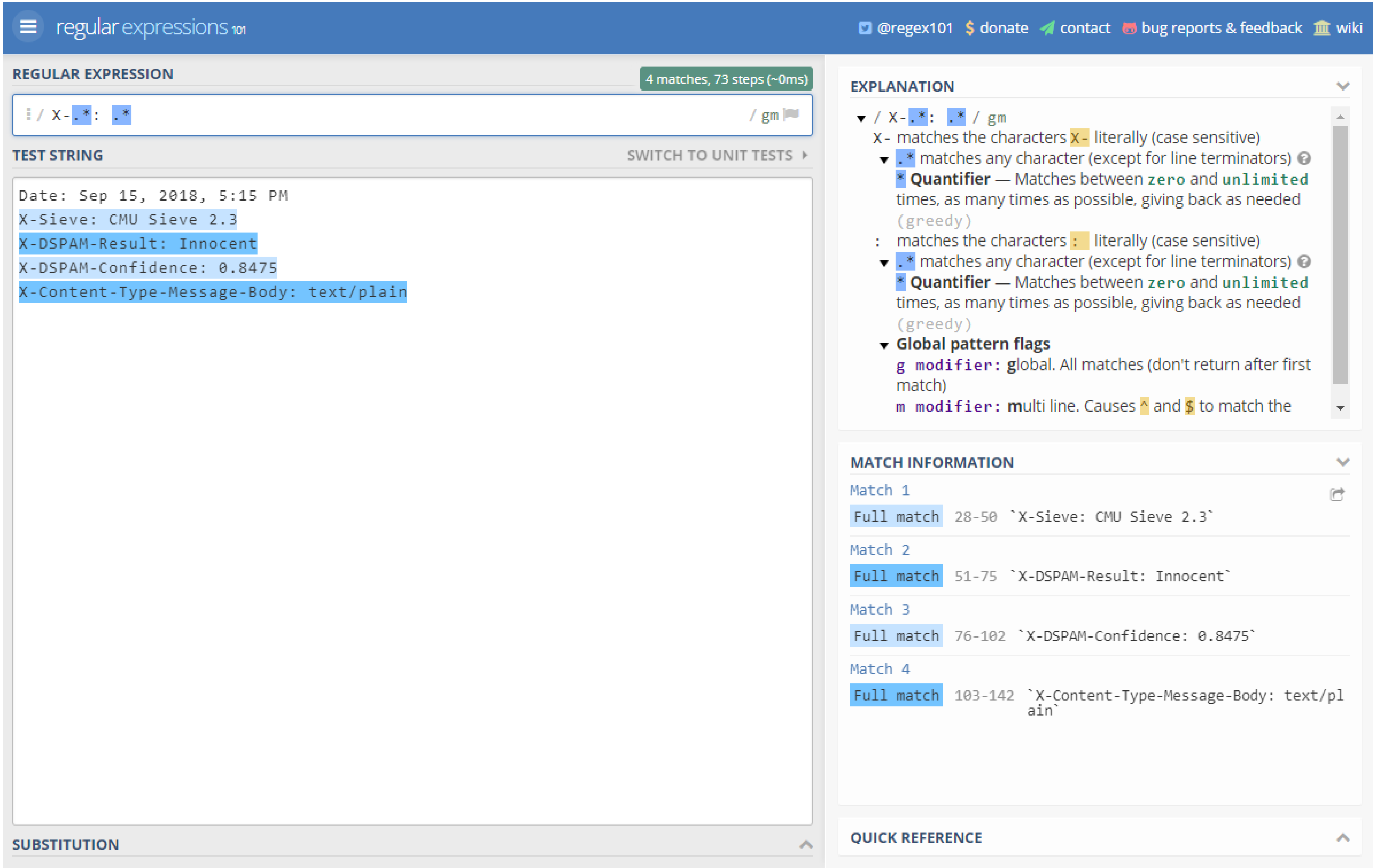
```
Date: Sep 15, 2018, 5:15 PM
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

- Let’s write a regex to just match just the X- ones:

```
/X-.*: .*/
```

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

Using Regex101.com to understand this



REGULAR EXPRESSION

4 matches, 73 steps (~0ms)

```
/ X-.*: .* / gm
```

TEST STRING

```
Date: Sep 15, 2018, 5:15 PM
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

EXPLANATION

/ X-.*: .* / gm

X- matches the characters X- literally (case sensitive)

.* matches any character (except for line terminators)

* **Quantifier** — Matches between `zero` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`

: matches the characters : literally (case sensitive)

.* matches any character (except for line terminators)

* **Quantifier** — Matches between `zero` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`

Global pattern flags

`g modifier:` global. All matches (don't return after first match)

`m modifier:` multi line. Causes ^ and $ to match the

MATCH INFORMATION

```
Match 1
Full match  28-50   `X-Sieve: CMU Sieve 2.3`
Match 2
Full match  51-75   `X-DSPAM-Result: Innocent`
Match 3
Full match  76-102  `X-DSPAM-Confidence: 0.8475`
Match 4
Full match  103-142 `X-Content-Type-Message-Body: text/plain`
```

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

Example: Email Headers
Capturing name and value

- We still have these email headers

```
Date: Sep 15, 2018, 5:15 PM
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

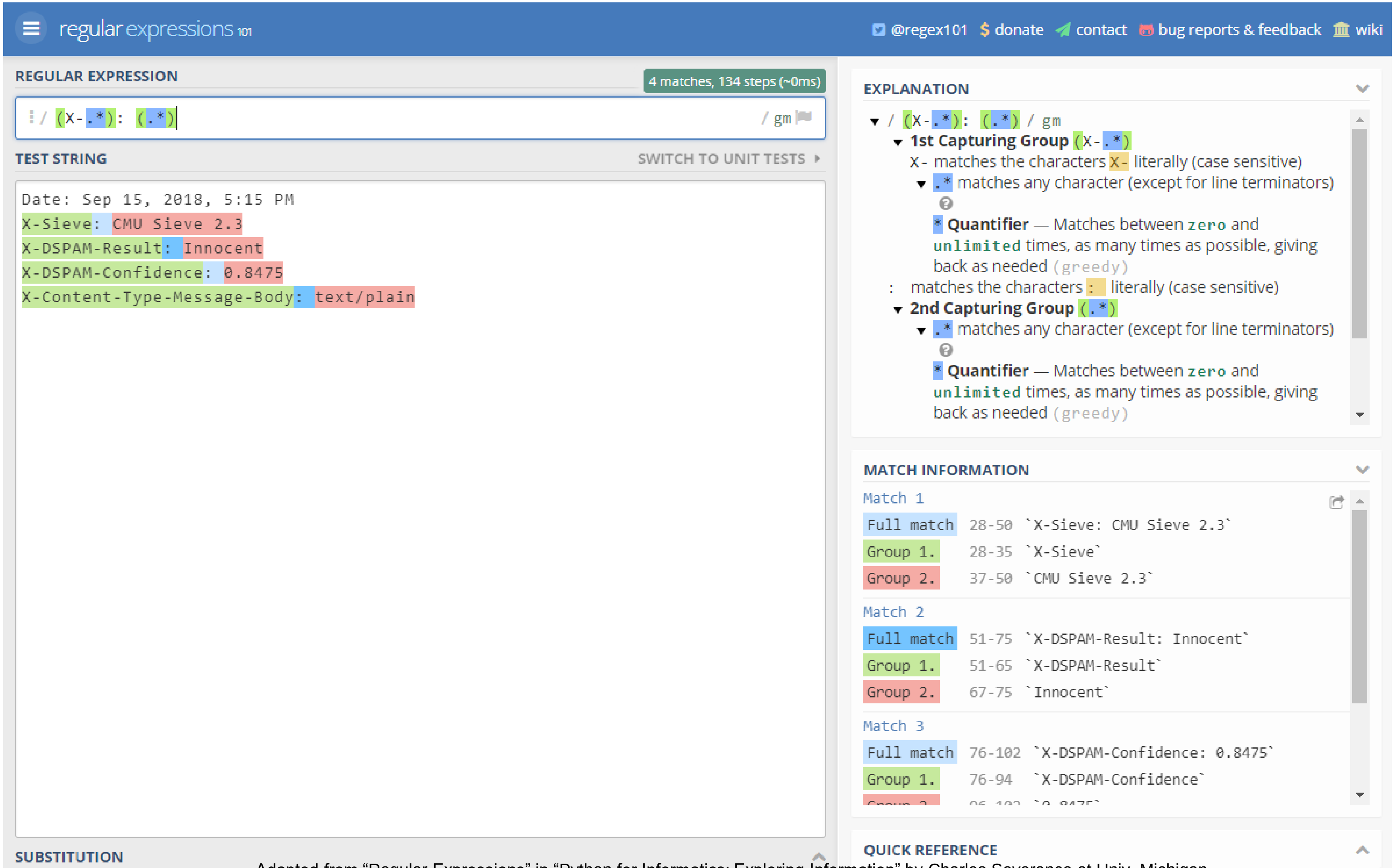
- Let’s amend our regex to capture the NAME and VALUE.

```
/(X-.*): (.*)/
```

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

What if we want to PARSE those headers?

- Parenthesis used for **capture** of part of a match



REGULAR EXPRESSION

4 matches, 134 steps (~0ms)

```
/ (X-.*): (.*) / gm
```

TEST STRING

```
Date: Sep 15, 2018, 5:15 PM
X-Sieve: CMU Sieve 2.3
X-DSPAM-Result: Innocent
X-DSPAM-Confidence: 0.8475
X-Content-Type-Message-Body: text/plain
```

EXPLANATION

- / (X-.*): (.*) / gm
  - 1st Capturing Group (X-.*)
    - X- matches the characters X- literally (case sensitive)
    - .* matches any character (except for line terminators)
      - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
  - : matches the characters : literally (case sensitive)
  - 2nd Capturing Group (.*)
    - .* matches any character (except for line terminators)
      - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

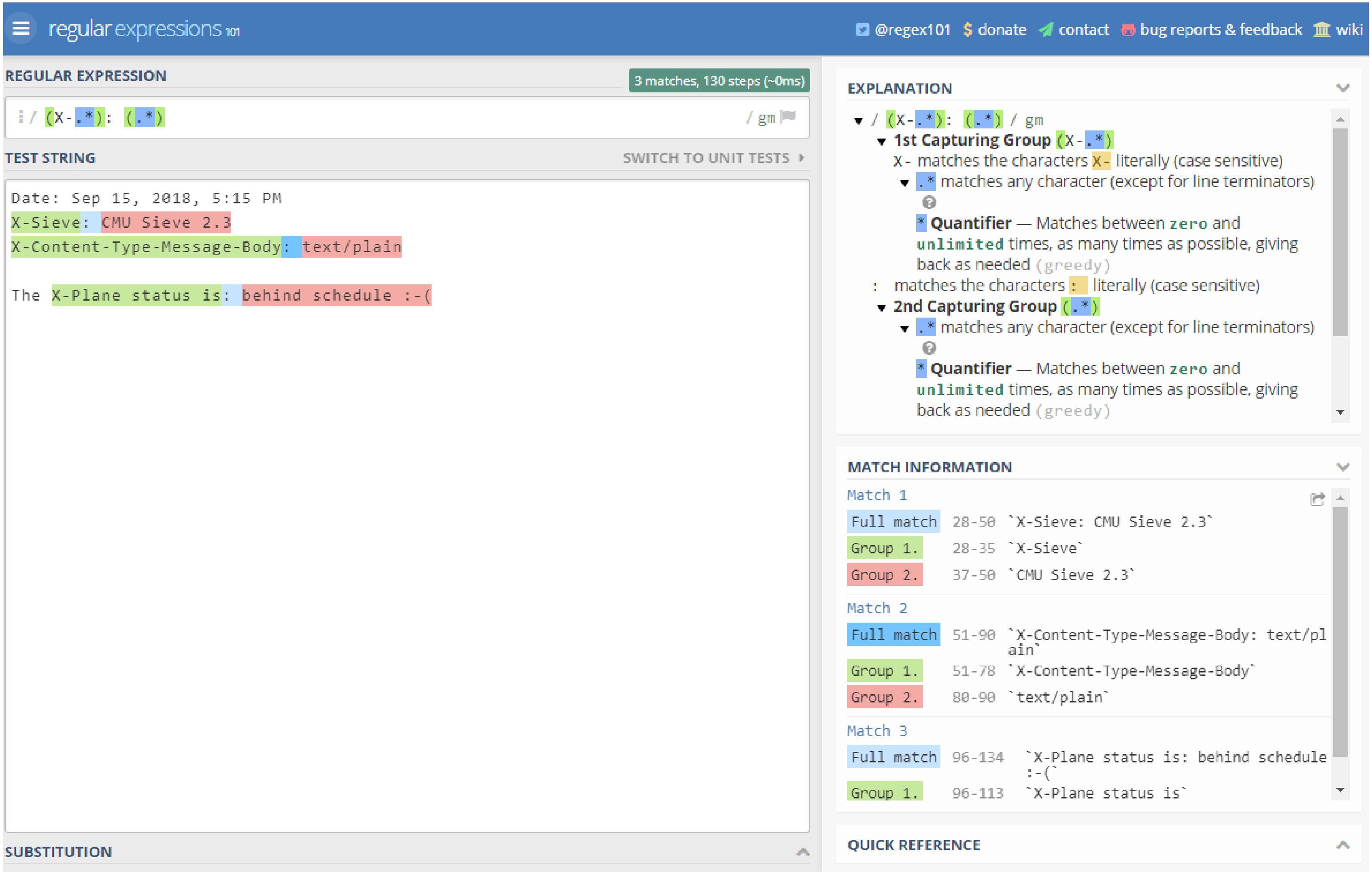
MATCH INFORMATION

```
Match 1
Full match   28-50    `X-Sieve: CMU Sieve 2.3`
Group 1.     28-35    `X-Sieve`
Group 2.     37-50    `CMU Sieve 2.3`
Match 2
Full match   51-75    `X-DSPAM-Result: Innocent`
Group 1.     51-65    `X-DSPAM-Result`
Group 2.     67-75    `Innocent`
Match 3
Full match   76-102   `X-DSPAM-Confidence: 0.8475`
Group 1.     76-94    `X-DSPAM-Confidence`
Group 2.     96-102   `0.8475`
```

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

Refining a regex (1)

- What if our content includes some confusing non-headers mixed in?



REGULAR EXPRESSION

3 matches, 130 steps (~0ms)

```
/ (X-.*): (.*) / gm
```

TEST STRING

SWITCH TO UNIT TESTS ▸

```
Date: Sep 15, 2018, 5:15 PM
X-Sieve: CMU Sieve 2.3
X-Content-Type-Message-Body: text/plain

The X-Plane status is: behind schedule :-(
```

SUBSTITUTION

EXPLANATION

▾ / (X-.*): (.*) / gm
- ▾ 1st Capturing Group (X-.*)
  - X- matches the characters X- literally (case sensitive)
  - ▾ .* matches any character (except for line terminators)
    - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
- : matches the characters : literally (case sensitive)
- ▾ 2nd Capturing Group (.*)
  - ▾ .* matches any character (except for line terminators)
    - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)

MATCH INFORMATION

```
Match 1
Full match   28-50   `X-Sieve: CMU Sieve 2.3`
Group 1.     28-35   `X-Sieve`
Group 2.     37-50   `CMU Sieve 2.3`

Match 2
Full match   51-90   `X-Content-Type-Message-Body: text/pl
                     ain`
Group 1.     51-78   `X-Content-Type-Message-Body`
Group 2.     80-90   `text/plain`

Match 3
Full match   96-134   `X-Plane status is: behind schedule
                      :-(`
Group 1.     96-113   `X-Plane status is`
```

QUICK REFERENCE

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

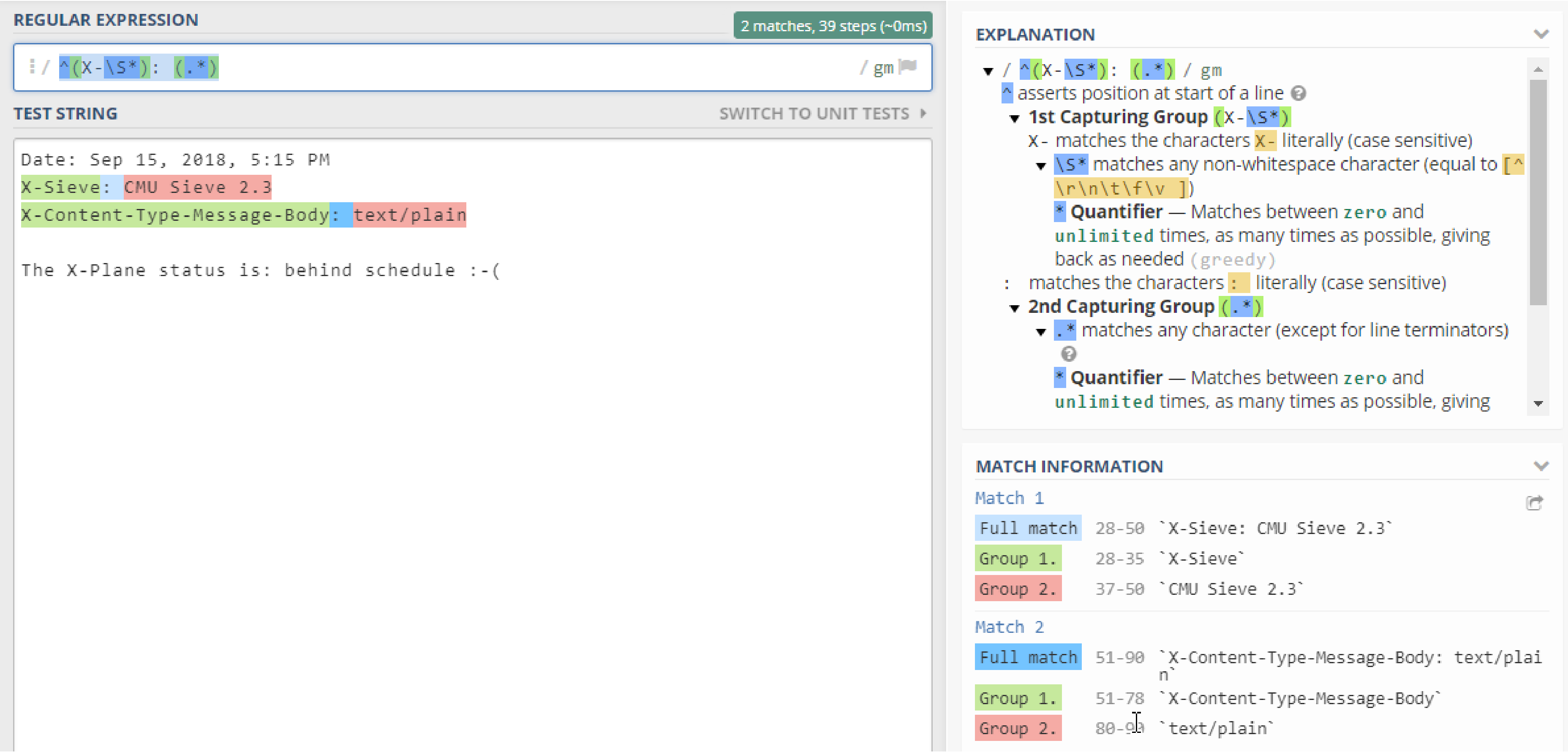
Refining a regex (2)

- Make regex more specific so it *just* matches what we want:

```
^(X-\S*): (.*)
```

Must be start of line

Non-whitespace characters only



REGULAR EXPRESSION

2 matches, 39 steps (~0ms)

```
/ ^(X-\S*): (.*) / gm
```

TEST STRING

SWITCH TO UNIT TESTS

```
Date: Sep 15, 2018, 5:15 PM
X-Sieve: CMU Sieve 2.3
X-Content-Type-Message-Body: text/plain

The X-Plane status is: behind schedule :-(
```

EXPLANATION

- / ^(X-\S*): (.*) / gm
  - ^ asserts position at start of a line
  - 1st Capturing Group (X-\S*)
    - X- matches the characters X- literally (case sensitive)
    - \S* matches any non-whitespace character (equal to [^\r\n\t\f\v])
      - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving back as needed (greedy)
  - : matches the characters : literally (case sensitive)
  - 2nd Capturing Group (.*)
    - .* matches any character (except for line terminators)
      - * Quantifier — Matches between zero and unlimited times, as many times as possible, giving

MATCH INFORMATION

```
Match 1
Full match   28-50   `X-Sieve: CMU Sieve 2.3`
Group 1.     28-35   `X-Sieve`
Group 2.     37-50   `CMU Sieve 2.3`

Match 2
Full match   51-90   `X-Content-Type-Message-Body: text/plai
                     n`
Group 1.     51-78   `X-Content-Type-Message-Body`
Group 2.     80-90   `text/plain`
```

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

Grouping *without* Backreferences

- Sometimes you just need to make a group
  - If important groups must be backreferenced, disable backreferencing for any unimportant groups
    - `$sentence =~ /(?:He|She) likes (\w+)\./;`
      - I don’t care if it’s a he or she
      - All I want to know is what he/she likes
      - Therefore I use `(?:)` to forgo the backreference
      - $1 will contain that thing that he/she likes

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Matching Modes

- Matching has different functional modes
  - In Perl, these are specified as letters after the regex.
    - `$name =~ /[a-z]+/i;`
      - `i` turns off case sensitivity
    - `$xml =~ /title=“([\w ]*)”.*keywords=“([\w ]*)”/s;`
      - `s` enables `.` to match newlines
    - `$report =~ /^\s*Name:[\s\S]*?The End.\s*$/m;`
      - `m` allows newlines between `^` and `$`
  - In Python, you pass an additional optional argument with named constants (either short like the above or with full names), e.g.:
    - `re.search(r'[a-z]+', name, re.I) # or re.IGNORECASE`

Regular Expression Substitution

- **Substitutions** simplify complex data modification
  - First part is a regex of what to find, second part is text to replace it
  - Backreferences can be included in replacement
  - For sophisticated work, most languages let you give a callback function so that the replacement can be programmatically generated for each match
- Perl replacement syntax
  - `$phone =~ s/\D//;`
    - Removes the first non-digit character in a phone number
      (Leaving the replacement blank means "replace with nothing", i.e. "delete")
  - `$html =~ s/^(\s*)/$1\t/;`
    - Adds a tab to a line of HTML using backreference `$1`
- Python uses `re.sub()`

Adapted from "Regular Expressions" by Ian Paterson at Rochester Institute of Technology

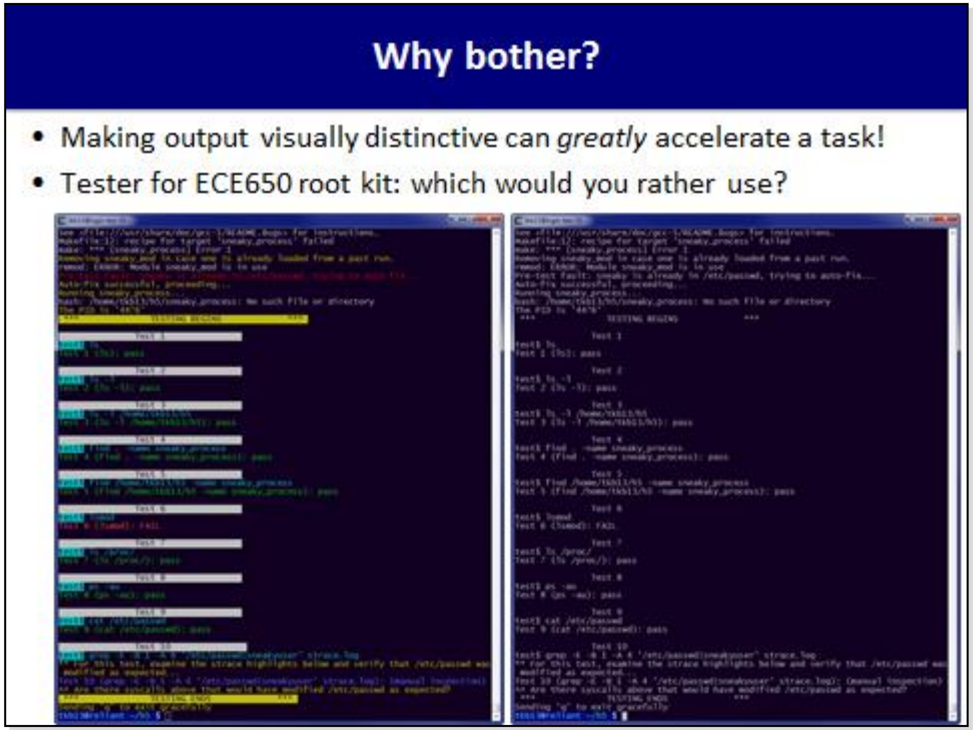
Substitutions Modes

- Substitutions have modes like matches (ignore case, multiline, etc.)
- Important one: Substitutions can be performed singly or **globally**
  - In Perl, use the **`g`** flag to force the expression to scan the entire string
    - **`$phone =~ s/\D//g;`**
      - Removes all non-digits in the phone number
  - In Python's **`re.sub()`** function, specify a **`count`** parameter to limit replacements (e.g. **`count=1`** for traditional "first match only" behavior)

Adapted from "Regular Expressions" by Ian Paterson at Rochester Institute of Technology

Combining one-liners and regexes

- Remember this slide when I compared color output to plain?



Why bother?

- Making output visually distinctive can *greatly* accelerate a task!
- Tester for ECE650 root kit: which would you rather use?

```
See <file:///usr/share/doc/gcc-5/README.Bugs> for instructions.
Makefile:12: recipe for target 'sneaky_process' failed
make: *** [sneaky_process] Error 1
Removing sneaky_mod in case one is already loaded from a past run.
rmmod: ERROR: Module sneaky_mod is in use
Pre-test fault: sneaky is already in /etc/passwd, trying to auto-fix...
Auto-fix successful, proceeding...
Running sneaky_process...
bash: /home/tkb13/h5/sneaky_process: No such file or directory
The PID is '4876'
 ***              TESTING BEGINS              ***

                  Test 1
test$ ls
Test 1 (ls): pass

                  Test 2
test$ ls -l
Test 2 (ls -l): pass

                  Test 3
test$ ls -l /home/tkb13/h5
Test 3 (ls -l /home/tkb13/h5)): pass

                  Test 4
test$ find . -name sneaky_process
Test 4 (find . -name sneaky_process)): pass

                  Test 5
test$ find /home/tkb13/h5 -name sneaky_process
Test 5 (find /home/tkb13/h5 -name sneaky_process)): pass

                  Test 6
test$ lsmod
Test 6 (lsmod): FAIL

                  Test 7
test$ ls /proc/
Test 7 (ls /proc/): pass

                  Test 8
test$ ps -au
Test 8 (ps -au): pass

                  Test 9
test$ cat /etc/passwd
Test 9 (cat /etc/passwd): pass

                  Test 10
test$ grep -E -B 1 -A 4 '/etc/passwd|sneakyuser' strace.log
** For this test, examine the strace highlights below and verify that /etc/passwd was
 modified as expected...
Test 10 (grep -E -B 1 -A 4 '/etc/passwd|sneakyuser' strace.log): (manual inspection)
** Are there syscalls above that would have modified /etc/passwd as expected?
 ***              TESTING ENDS              ***
Sending 'q' to exit gracefully
tkb13@reliant ~/h5 $
```

- Did I write a whole separate script that omitted colors? NO!

```
$ perl -pe 's/\e.*?m//g' orig > plain
```

  - **-p** means “read one line at a time like -n, but print the line afterwards”
  - **\e** is the escape character.

Regular Expression Quick Guide

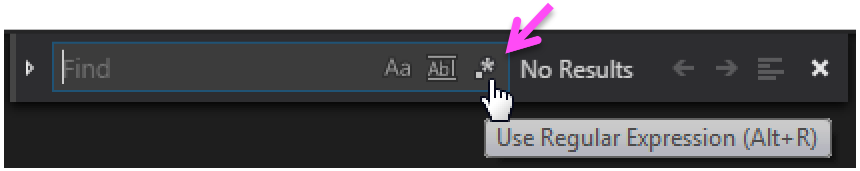
`^`	Matches the beginning of a line
`$`	Matches the end of the line
`.`	Matches any character (except newline, unless you give an option)
`\s`	Matches whitespace
`\S`	Matches any non-whitespace character
`\w`	Matches a “word-like” character (letters/numbers/underscore)
`\d`	Matches a decimal digit (0-9)
`\b`	Matches a word boundary
`?`	Makes a character or group *optional* (appears zero or one times)
`*`	Repeats a character or group zero or more times
`*?`	Repeats a character zero or more times (non-greedy)
`+`	Repeats a character one or more times
`+?`	Repeats a character one or more times (non-greedy)
`|`	Alternation – allows either/or. Usually used with parens: `(this)|(that)`
`[aeiou]`	Matches a single character in the listed set
`[^XYZ]`	Matches a single character not in the listed set
`[a-z0-9]`	The set of characters can include a range
`(` and `)`	Indicates a group (used to capture part of a match *or* group stuff for modifiers)

A more complete quick-ref guide is here and linked on the course site. See also the Python `re` module docs.

Adapted from “Regular Expressions” in “Python for Informatics: Exploring Information” by Charles Severance at Univ. Michigan

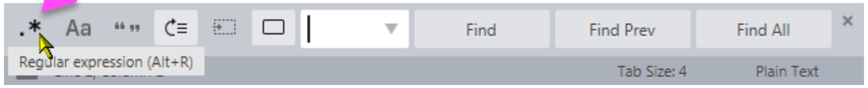
Where can I use regexes?

- Obviously, in Perl and Python
- Also: Javascript, Java, .NET, PHP, R, C/C++, PowerShell, Ruby
- Also: your text editor



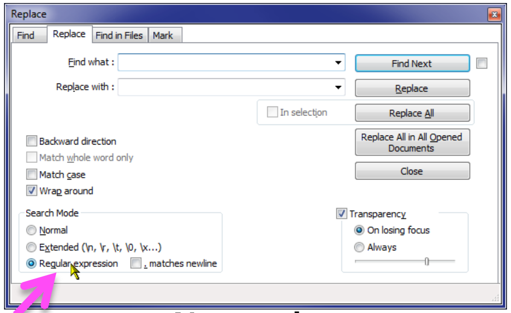
Use Regular Expression (Alt+R)

Microsoft VS Code



Regular expression (Alt+R)

Sublime Text



Find what :

Replace with :

□ In selection

□ Backward direction
□ Match whole word only
□ Match case
☑ Wrap around

Search Mode
○ Normal
○ Extended (\n, \r, \t, \0, \x...)
● Regular expression □ . matches newline

☑ Transparency
● On losing focus
○ Always

Notepad++

No screenshot because I ain’t launching that thing but you can type “C-M-s” for regex search (whatever that means)

emacs

```
/(regex) goes [here]
```

vi and vim (press /)

- Tons of shell tools:
  - `grep -P`
  - `sed` (just has em)
  - `awk` (just has em)
  - `less` (Press /)

- **Everyone cool is using regexes! Don’t get left behind!!!!**

References for learning more about regexs

- Regex editor, code generator, and community database of regexs
  - http://regex101.com/

- Tutorials for various programming languages
  - http://www.regular-expressions.info/
- Python in-depth docs
  - https://docs.python.org/3/library/re.html
- Perl in-depth docs
  - https://perldoc.perl.org/perlreref.html

Adapted from “Regular Expressions” by Ian Paterson at Rochester Institute of Technology

Manipulating tabular data

Hey, how about Excel? That things cool, right?

- Terminals are nice, but did you know: GUIs exist?
- Some tasks benefit from non-terminal interface
- Example: tabular data wants to be in a **spreadsheet**

- Let’s cover some quick tips on (ab)using Excel (or Google Sheets)

Data in/out

- File format for getting in/out of Excel: ***Comma-Separated Values (CSV)***
  - Trivial for simple data: bob,2,19
  - If you have commas in data, enclose in quotes: "Jimmy, PhD",4,50
  - If you have quotes, double them up: "This is a ""quote""",7,94
  - Save with ".csv" extension an Excel loads it right up
  - Can generate well enough with simple commands
  - Can use common libraries to do everything "right" (quoting, etc.); e.g. Python has a built-in **`csv`** module
- For fast stuff, can just use the **clipboard**
  - Often quick just to copy/paste instead of making actual files
  - Format for spreadsheet <-> plaintext via clipboard is **tab-separated**
  - For a single column of data, there's no tabs – it's just in lines!

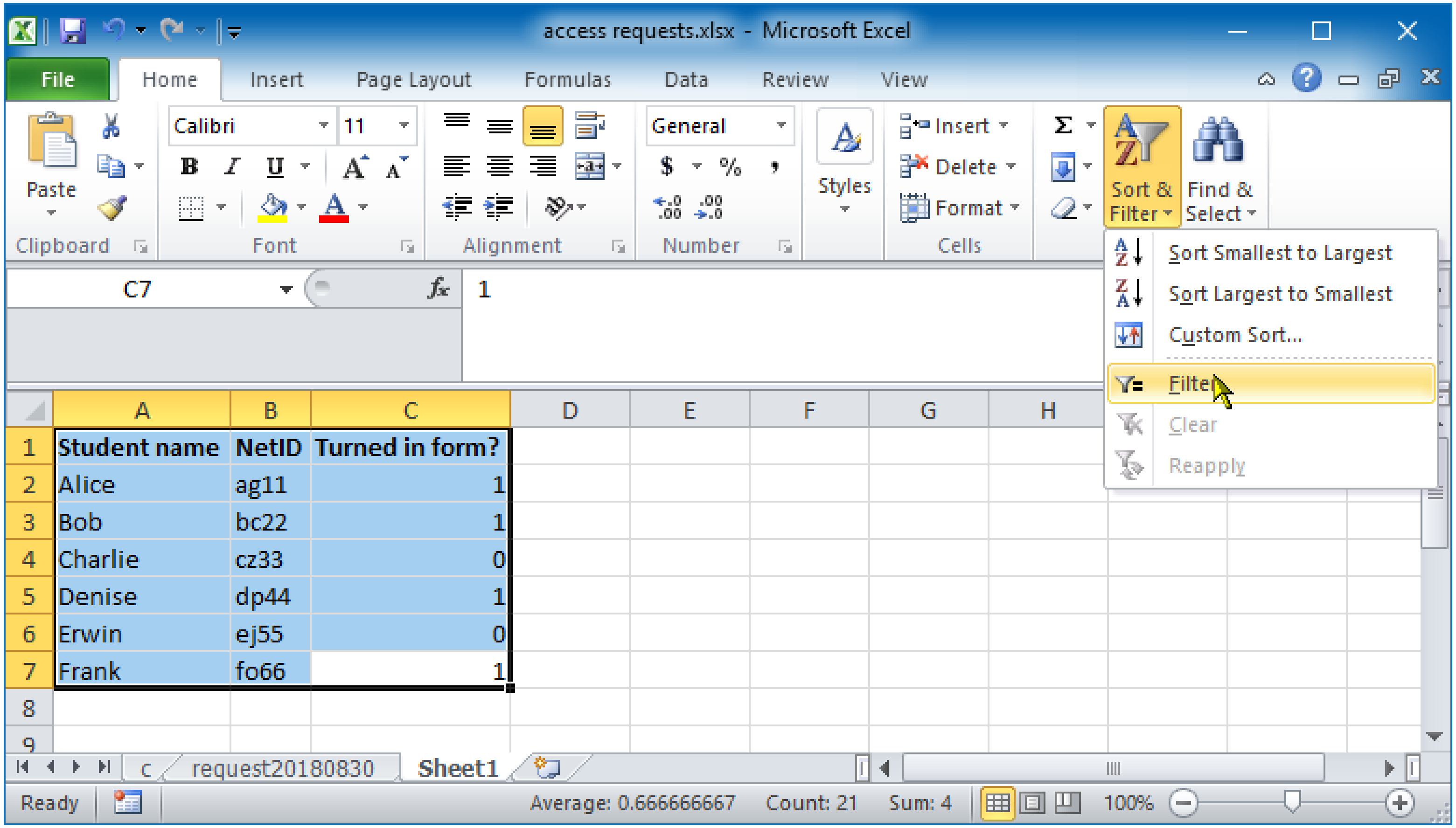
Formulas

- Spreadsheet formulas are outside of our scope – if you aren’t familiar, you **need** to learn them

- One thing to add: you can do string manipulation as well as math
  - & is the concatenation operator
  - TEXT() can format numbers in arbitrary formats

Auto-filter

- Take a sheet, make sure it has headers, highlight your data, turn on **auto filter** → Bam! **instant sort/filter controls**.
  - Example: Requesting badge access for some students.



access requests.xlsx - Microsoft Excel

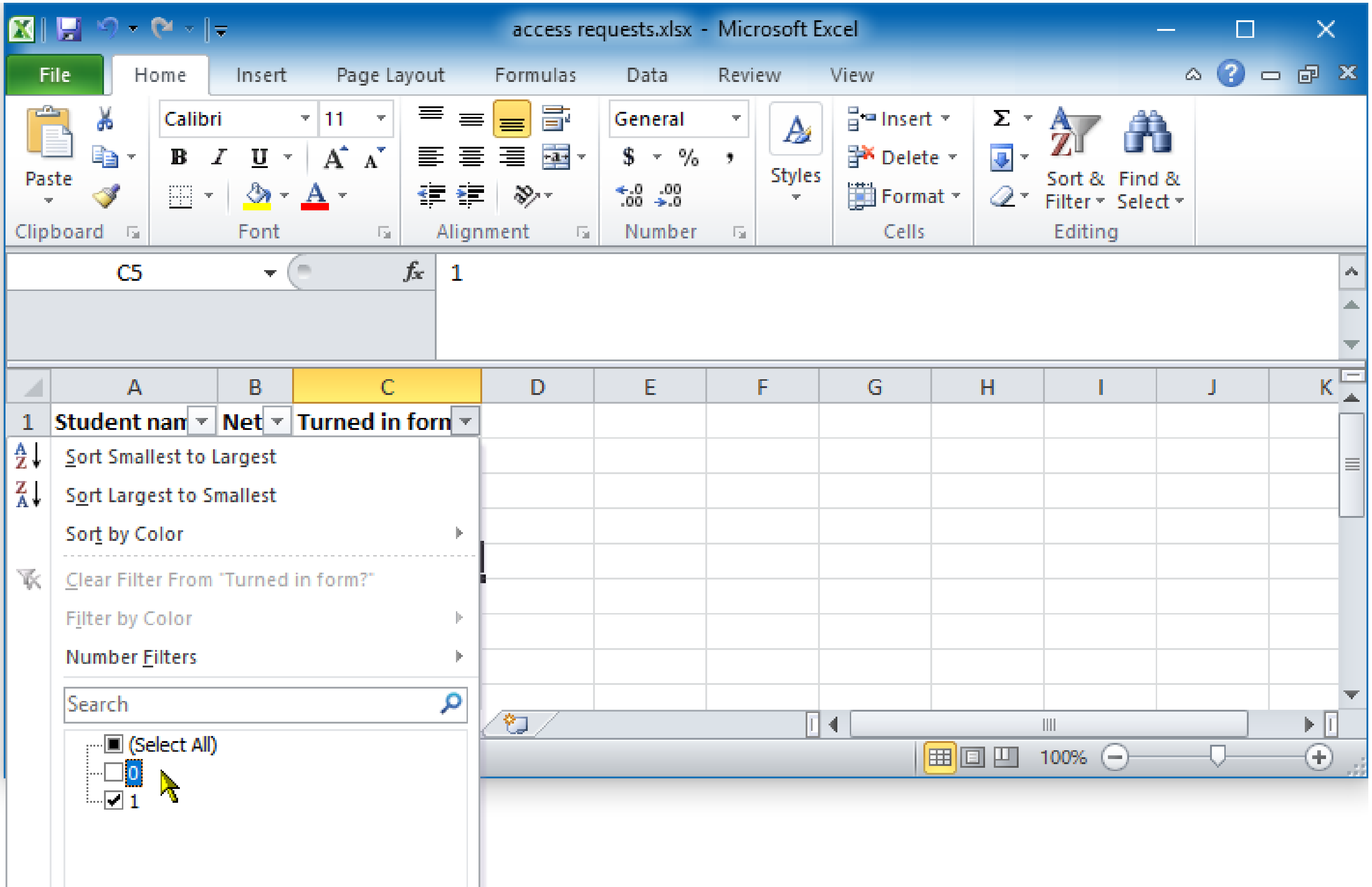
C7 = 1

	A	B	C
1	Student name	NetID	Turned in form?
2	Alice	ag11	1
3	Bob	bc22	1
4	Charlie	cz33	0
5	Denise	dp44	1
6	Erwin	ej55	0
7	Frank	fo66	1

Average: 0.666666667 Count: 21 Sum: 4

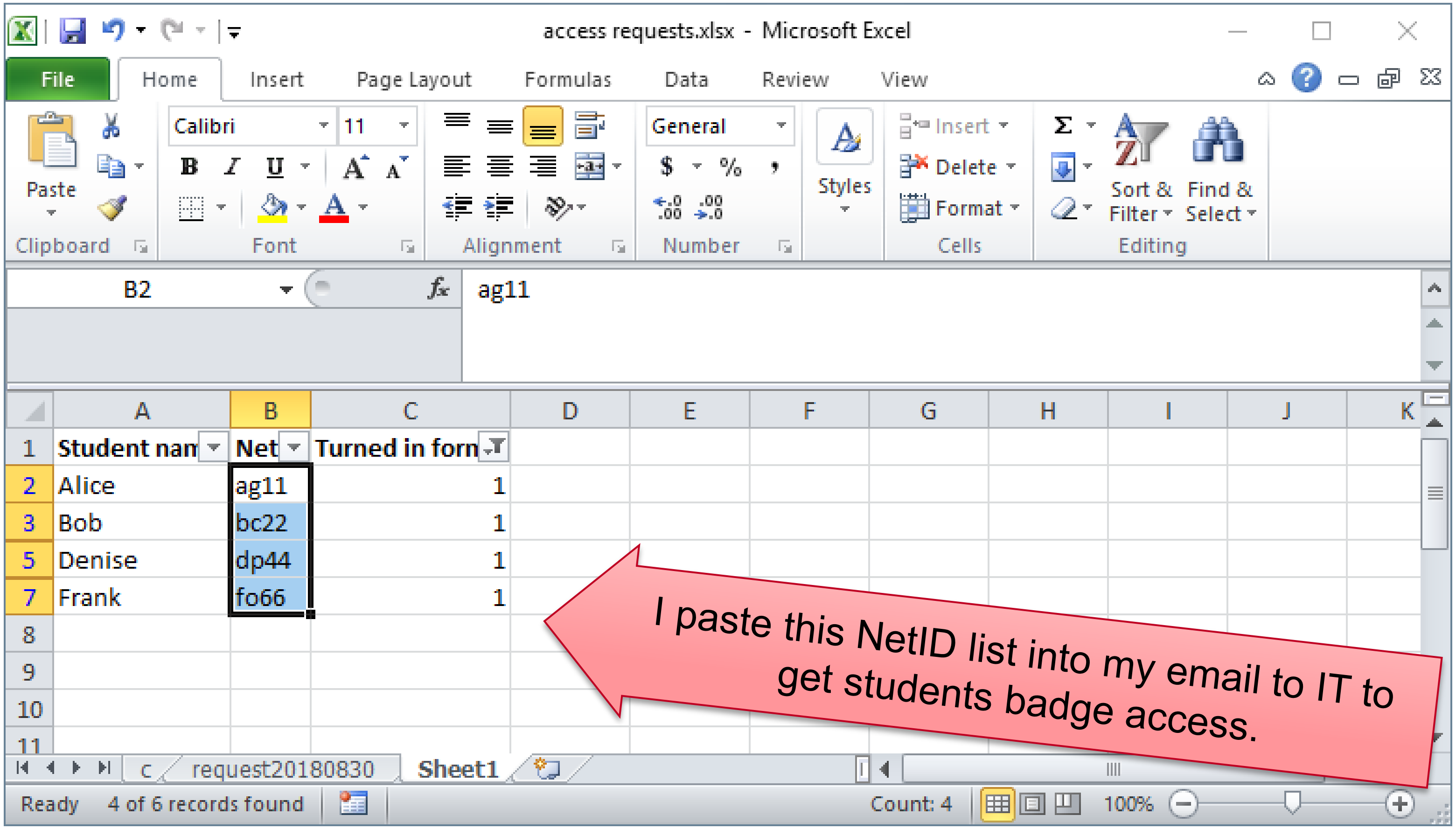
Auto-filter

- Take a sheet, make sure it has headers, highlight your data, turn on **auto filter** → Bam! **instant sort/filter controls**.
  - Example: Requesting badge access for some students.



Auto-filter

- Take a sheet, make sure it has headers, highlight your data, turn on **auto filter** → Bam! **instant sort/filter controls**.
  - Example: Requesting badge access for some students.



B2 | ag11

	A	B	C
1	Student nam	Net	Turned in forn
2	Alice	ag11	1
3	Bob	bc22	1
5	Denise	dp44	1
7	Frank	fo66	1

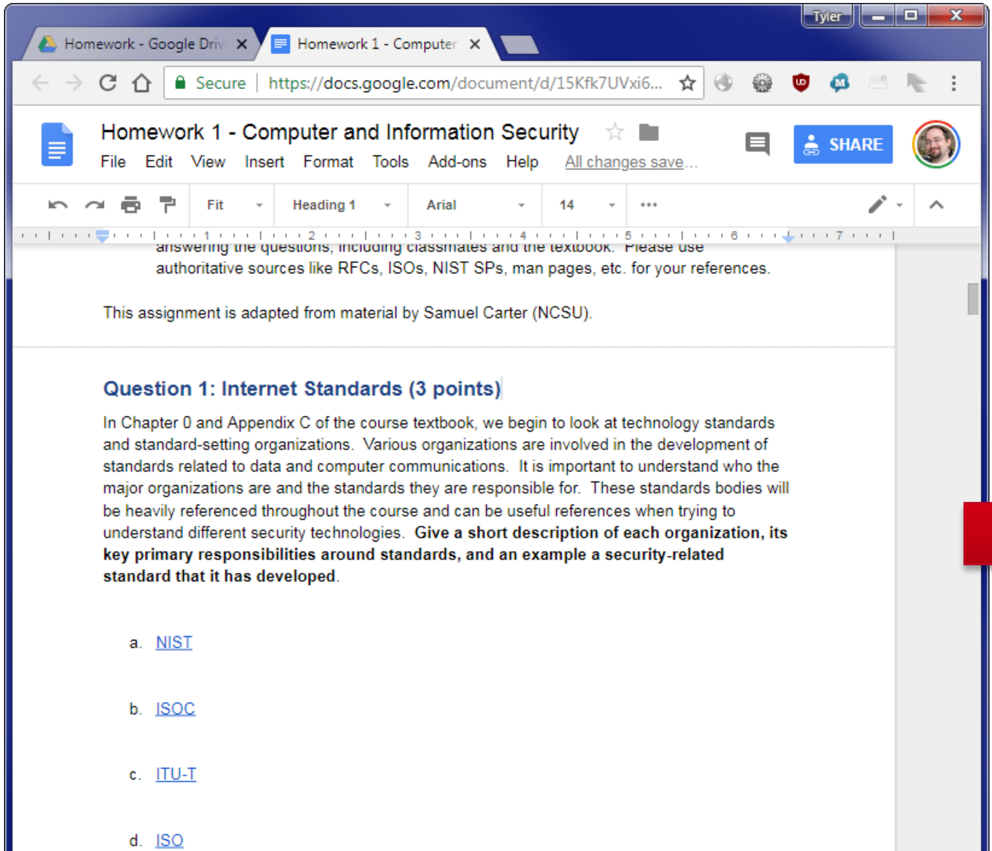
Ready 4 of 6 records found Count: 4

I paste this NetID list into my email to IT to get students badge access.

Putting it all together

Example data manipulation task: Planning Homework 1

- **What I have**: The Homework 1 draft writeup
- **My goal**: Plan out point allocation for questions
- **What I want**: Table of question number, topic, and points



Homework - Google Driv
Homework 1 - Computer
Tyler
Secure
https://docs.google.com/document/d/15Kfk7UVxi6...
Homework 1 - Computer and Information Security
File Edit View Insert Format Tools Add-ons Help
All changes save...
SHARE
Fit
Heading 1
Arial
14

answering the questions, including classmates and the textbook. Please use authoritative sources like RFCs, ISOs, NIST SPs, man pages, etc. for your references.

This assignment is adapted from material by Samuel Carter (NCSU).

Question 1: Internet Standards (3 points)

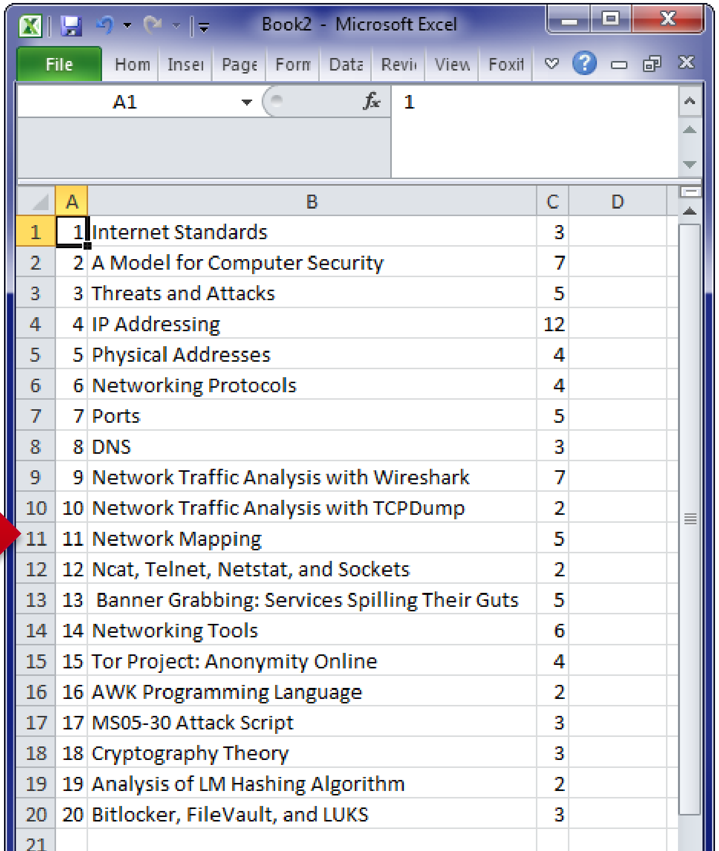
In Chapter 0 and Appendix C of the course textbook, we begin to look at technology standards and standard-setting organizations. Various organizations are involved in the development of standards related to data and computer communications. It is important to understand who the major organizations are and the standards they are responsible for. These standards bodies will be heavily referenced throughout the course and can be useful references when trying to understand different security technologies. **Give a short description of each organization, its key primary responsibilities around standards, and an example a security-related standard that it has developed.**

a. NIST

b. ISOC

c. ITU-T

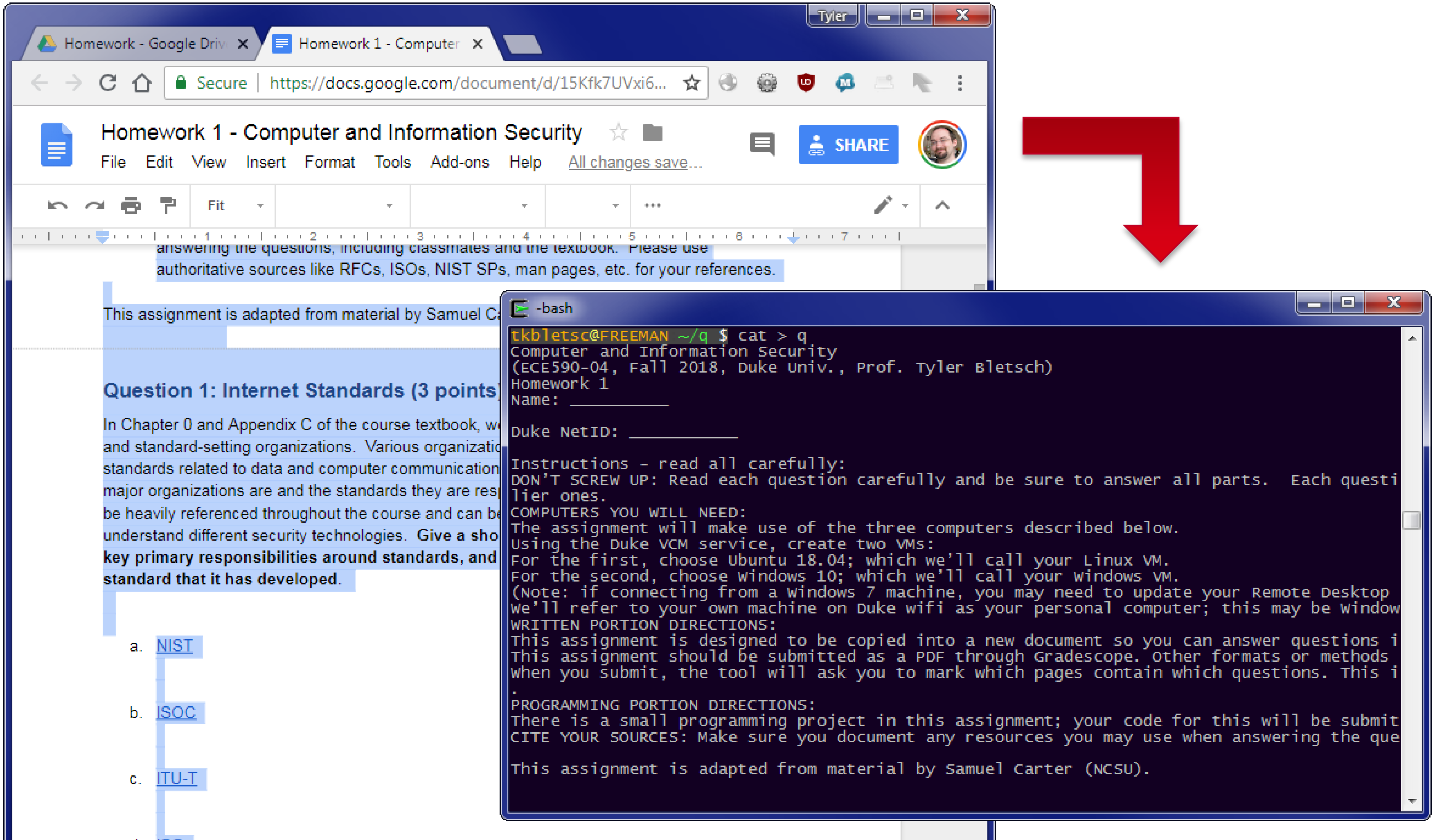
d. ISO



	A	B	C	D
1	1	Internet Standards	3	
2	2	A Model for Computer Security	7	
3	3	Threats and Attacks	5	
4	4	IP Addressing	12	
5	5	Physical Addresses	4	
6	6	Networking Protocols	4	
7	7	Ports	5	
8	8	DNS	3	
9	9	Network Traffic Analysis with Wireshark	7	
10	10	Network Traffic Analysis with TCPDump	2	
11	11	Network Mapping	5	
12	12	Ncat, Telnet, Netstat, and Sockets	2	
13	13	Banner Grabbing: Services Spilling Their Guts	5	
14	14	Networking Tools	6	
15	15	Tor Project: Anonymity Online	4	
16	16	AWK Programming Language	2	
17	17	MS05-30 Attack Script	3	
18	18	Cryptography Theory	3	
19	19	Analysis of LM Hashing Algorithm	2	
20	20	Bitlocker, FileVault, and LUKS	3	
21				

Planning Homework 1: Acquire source data

- Select all, copy
- In shell, run "`cat > q`" and paste (middle-click), then **Ctrl+D** for EOF



answering the questions, including classmates and the textbook. Please use authoritative sources like RFCs, ISOs, NIST SPs, man pages, etc. for your references.

This assignment is adapted from material by Samuel C

Question 1: Internet Standards (3 points

In Chapter 0 and Appendix C of the course textbook, w
and standard-setting organizations. Various organizati
standards related to data and computer communicatio
major organizations are and the standards they are res
be heavily referenced throughout the course and can b
understand different security technologies. **Give a sho
key primary responsibilities around standards, and
standard that it has developed.**

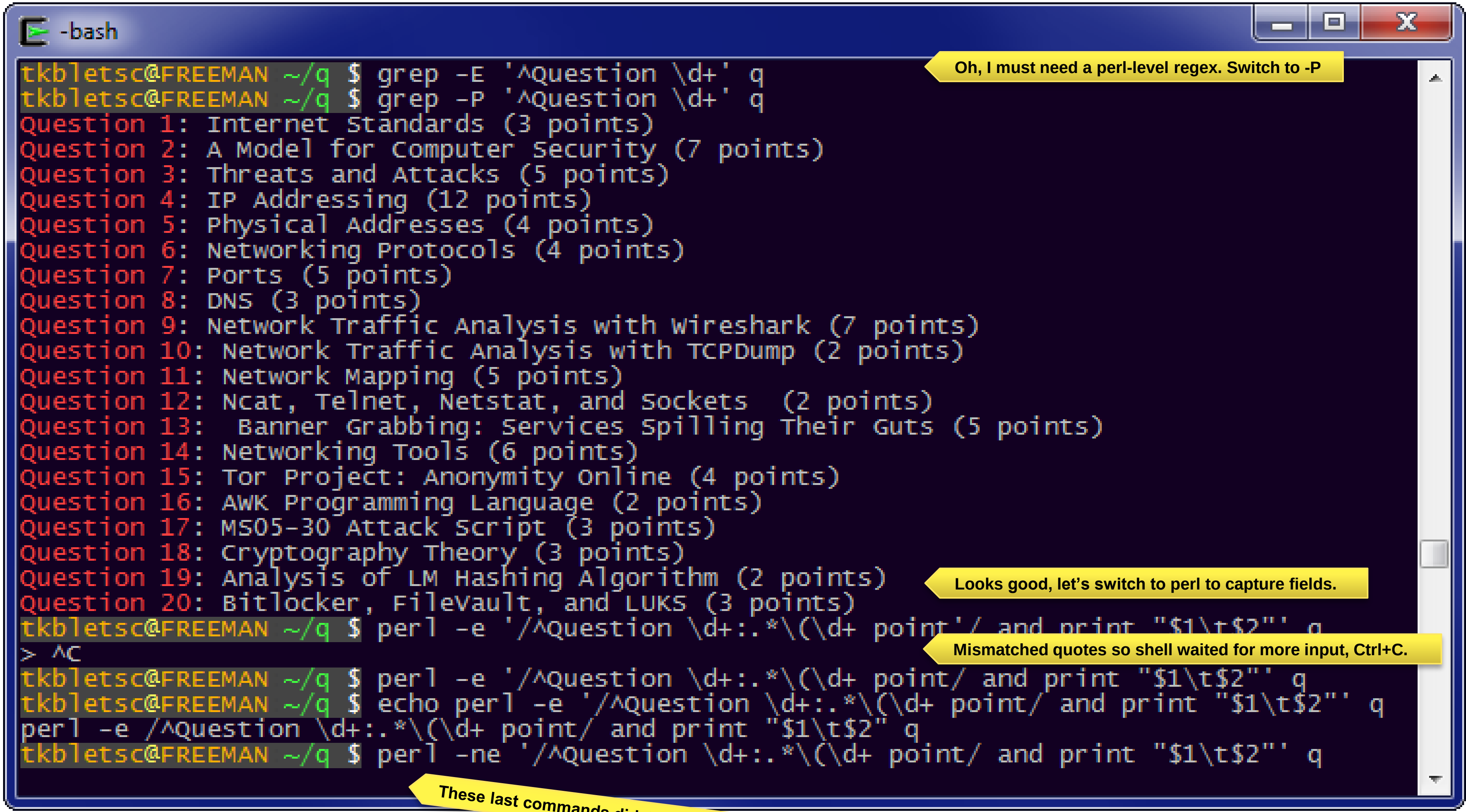
a. NIST

b. ISOC

c. ITU-T

```
tkbletsc@FREEMAN ~/q $ cat > q
Computer and Information Security
(ECE590-04, Fall 2018, Duke Univ., Prof. Tyler Bletsch)
Homework 1
Name: __________

Duke NetID: ___________

Instructions - read all carefully:
DON'T SCREW UP: Read each question carefully and be sure to answer all parts.  Each questi
lier ones.
COMPUTERS YOU WILL NEED:
The assignment will make use of the three computers described below.
Using the Duke VCM service, create two VMs:
For the first, choose Ubuntu 18.04; which we'll call your Linux VM.
For the second, choose Windows 10; which we'll call your Windows VM.
(Note: if connecting from a Windows 7 machine, you may need to update your Remote Desktop
We'll refer to your own machine on Duke wifi as your personal computer; this may be Window
WRITTEN PORTION DIRECTIONS:
This assignment is designed to be copied into a new document so you can answer questions i
This assignment should be submitted as a PDF through Gradescope. Other formats or methods
When you submit, the tool will ask you to mark which pages contain which questions. This i
.
PROGRAMMING PORTION DIRECTIONS:
There is a small programming project in this assignment; your code for this will be submit
CITE YOUR SOURCES: Make sure you document any resources you may use when answering the que

This assignment is adapted from material by Samuel Carter (NCSU).
```

Planning Homework 1:
Develop regex



```
tkbletsc@FREEMAN ~/q $ grep -E '^Question \d+' q
tkbletsc@FREEMAN ~/q $ grep -P '^Question \d+' q
Question 1: Internet Standards (3 points)
Question 2: A Model for Computer Security (7 points)
Question 3: Threats and Attacks (5 points)
Question 4: IP Addressing (12 points)
Question 5: Physical Addresses (4 points)
Question 6: Networking Protocols (4 points)
Question 7: Ports (5 points)
Question 8: DNS (3 points)
Question 9: Network Traffic Analysis with Wireshark (7 points)
Question 10: Network Traffic Analysis with TCPDump (2 points)
Question 11: Network Mapping (5 points)
Question 12: Ncat, Telnet, Netstat, and Sockets  (2 points)
Question 13:  Banner Grabbing: Services Spilling Their Guts (5 points)
Question 14: Networking Tools (6 points)
Question 15: Tor Project: Anonymity Online (4 points)
Question 16: AWK Programming Language (2 points)
Question 17: MS05-30 Attack Script (3 points)
Question 18: Cryptography Theory (3 points)
Question 19: Analysis of LM Hashing Algorithm (2 points)
Question 20: Bitlocker, FileVault, and LUKS (3 points)
tkbletsc@FREEMAN ~/q $ perl -e '/^Question \d+:.*\(\d+ point'/ and print "$1\t$2"' q
> ^C
tkbletsc@FREEMAN ~/q $ perl -e '/^Question \d+:.*\(\d+ point/ and print "$1\t$2"' q
tkbletsc@FREEMAN ~/q $ echo perl -e '/^Question \d+:.*\(\d+ point/ and print "$1\t$2"' q
perl -e /^Question \d+:.*\(\d+ point/ and print "$1\t$2" q
tkbletsc@FREEMAN ~/q $ perl -ne '/^Question \d+:.*\(\d+ point/ and print "$1\t$2"' q
```

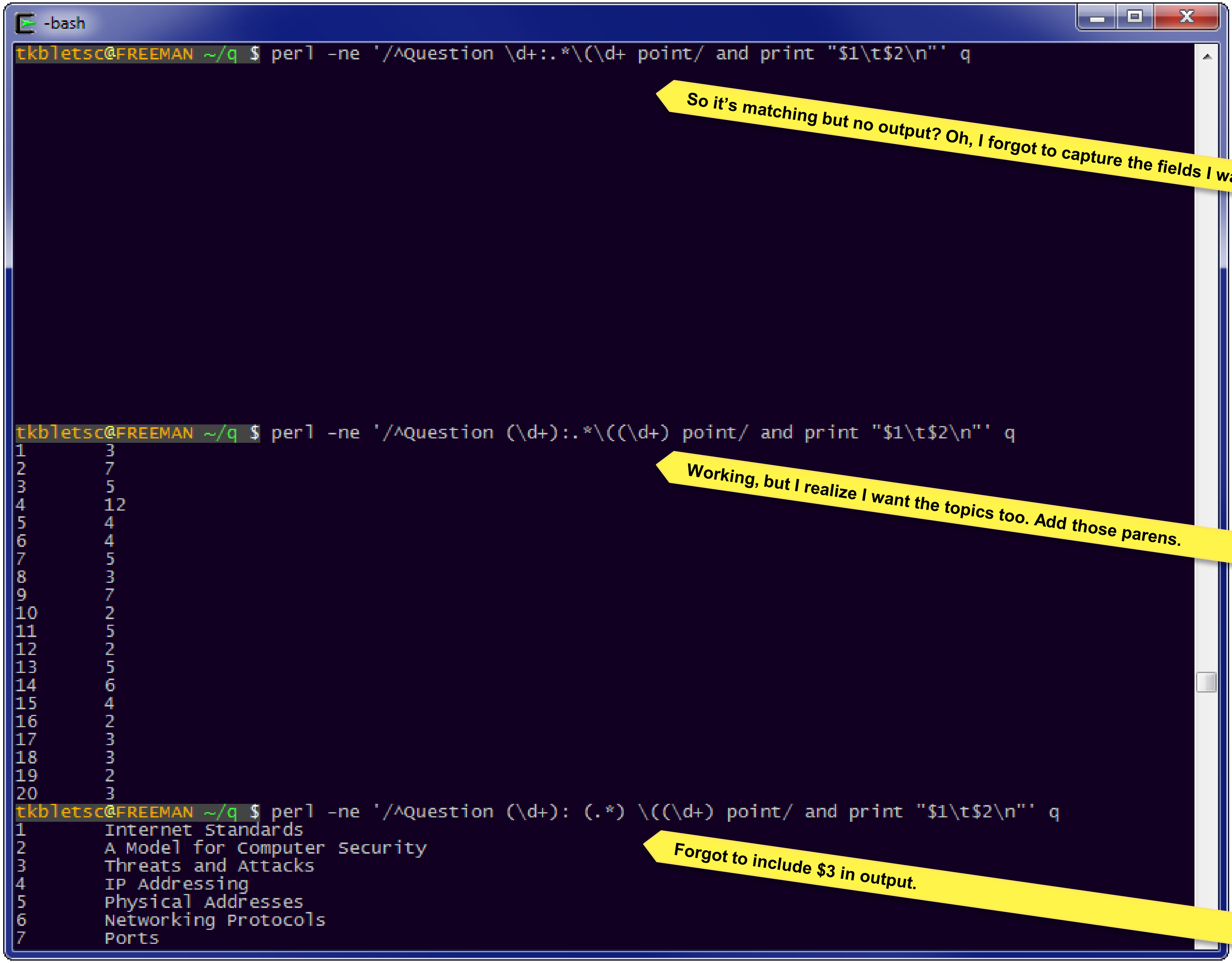
Oh, I must need a perl-level regex. Switch to -P

Looks good, let's switch to perl to capture fields.

Mismatched quotes so shell waited for more input, Ctrl+C.

These last commands didn't match anything – I realize I forgot to turn on 'read a file line by line' mode (-n)

Planning Homework 1:
Debug regex



```
tkbletsc@FREEMAN ~/q $ perl -ne '/^Question \d+:.*\(\d+ point/ and print "$1\t$2\n"' q
```

So it’s matching but no output? Oh, I forgot to capture the fields I wanted! Add capture parens.

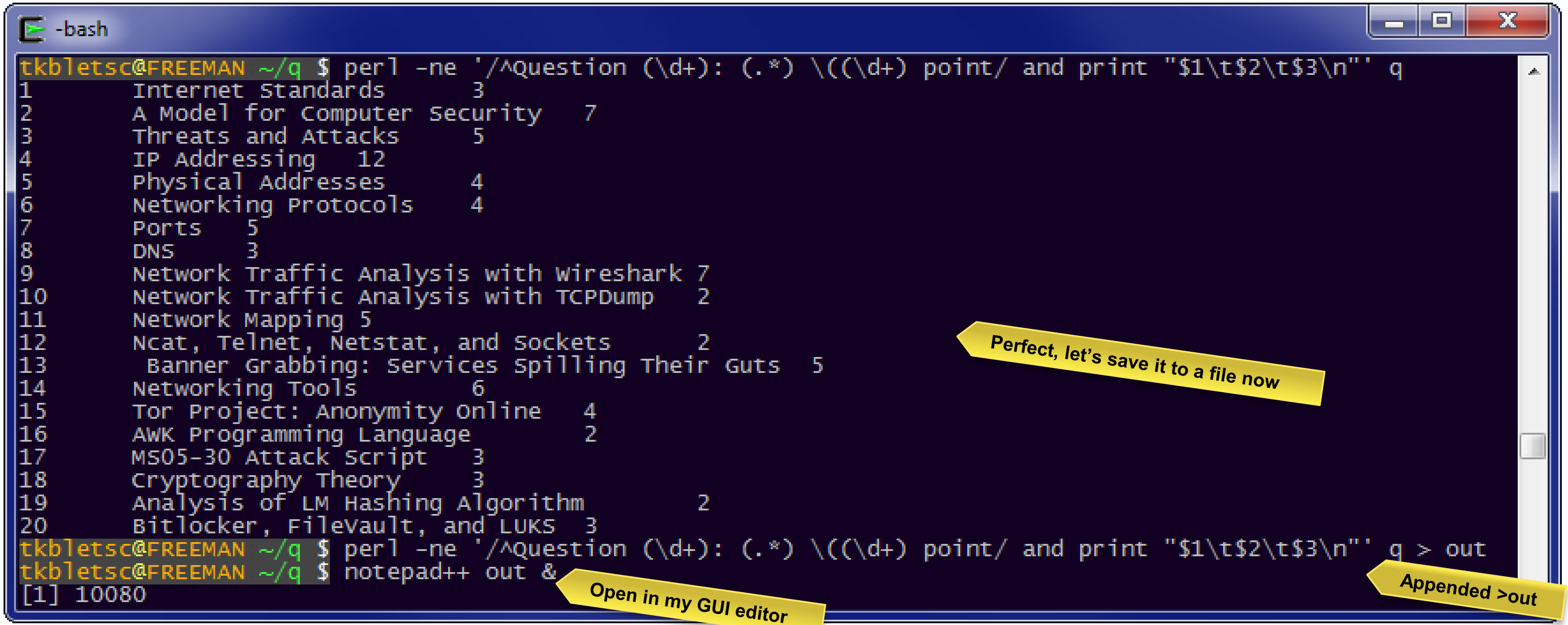
```
tkbletsc@FREEMAN ~/q $ perl -ne '/^Question (\d+):.*\((\d+) point/ and print "$1\t$2\n"' q
1       3
2       7
3       5
4       12
5       4
6       4
7       5
8       3
9       7
10      2
11      5
12      2
13      5
14      6
15      4
16      2
17      3
18      3
19      2
20      3
```

Working, but I realize I want the topics too. Add those parens.

```
tkbletsc@FREEMAN ~/q $ perl -ne '/^Question (\d+): (.*) \((\d+) point/ and print "$1\t$2\n"' q
1       Internet Standards
2       A Model for Computer Security
3       Threats and Attacks
4       IP Addressing
5       Physical Addresses
6       Networking Protocols
7       Ports
```

Forgot to include $3 in output.

Planning Homework 1: Clean output

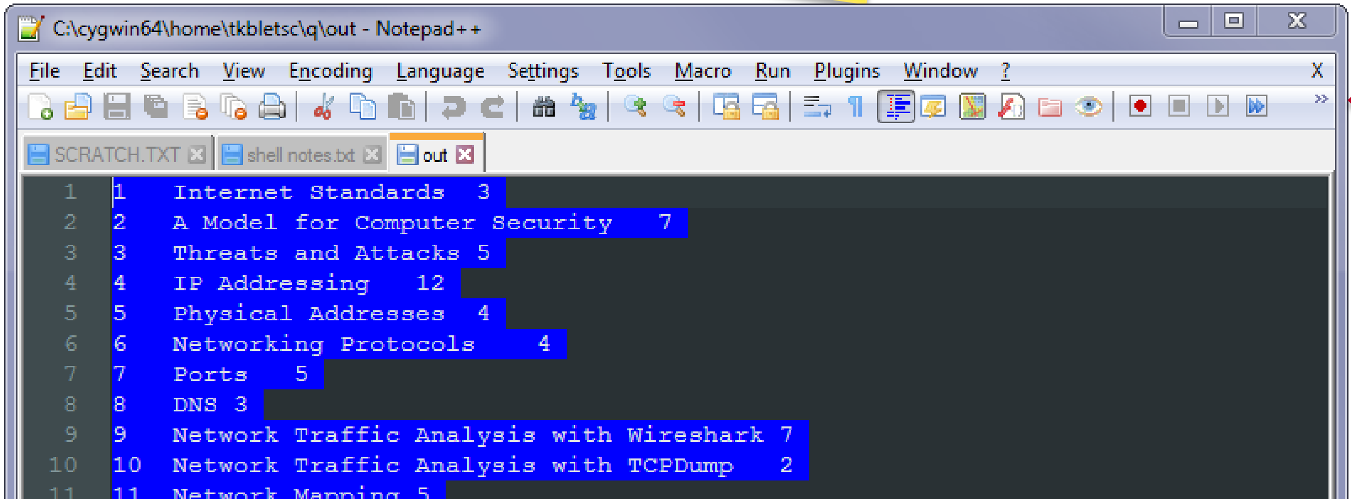


```
tkbletsc@FREEMAN ~/q $ perl -ne '/^Question (\d+): (.*) \((\d+) point/ and print "$1\t$2\t$3\n"' q
1       Internet Standards      3
2       A Model for Computer Security   7
3       Threats and Attacks     5
4       IP Addressing   12
5       Physical Addresses      4
6       Networking Protocols    4
7       Ports   5
8       DNS     3
9       Network Traffic Analysis with Wireshark 7
10      Network Traffic Analysis with TCPDump   2
11      Network Mapping 5
12      Ncat, Telnet, Netstat, and Sockets      2
13       Banner Grabbing: Services Spilling Their Guts  5
14      Networking Tools        6
15      Tor Project: Anonymity Online   4
16      AWK Programming Language        2
17      MS05-30 Attack Script   3
18      Cryptography Theory     3
19      Analysis of LM Hashing Algorithm        2
20      Bitlocker, FileVault, and LUKS  3
tkbletsc@FREEMAN ~/q $ perl -ne '/^Question (\d+): (.*) \((\d+) point/ and print "$1\t$2\t$3\n"' q > out
tkbletsc@FREEMAN ~/q $ notepad++ out &
[1] 10080
```

Perfect, let's save it to a file now

Open in my GUI editor

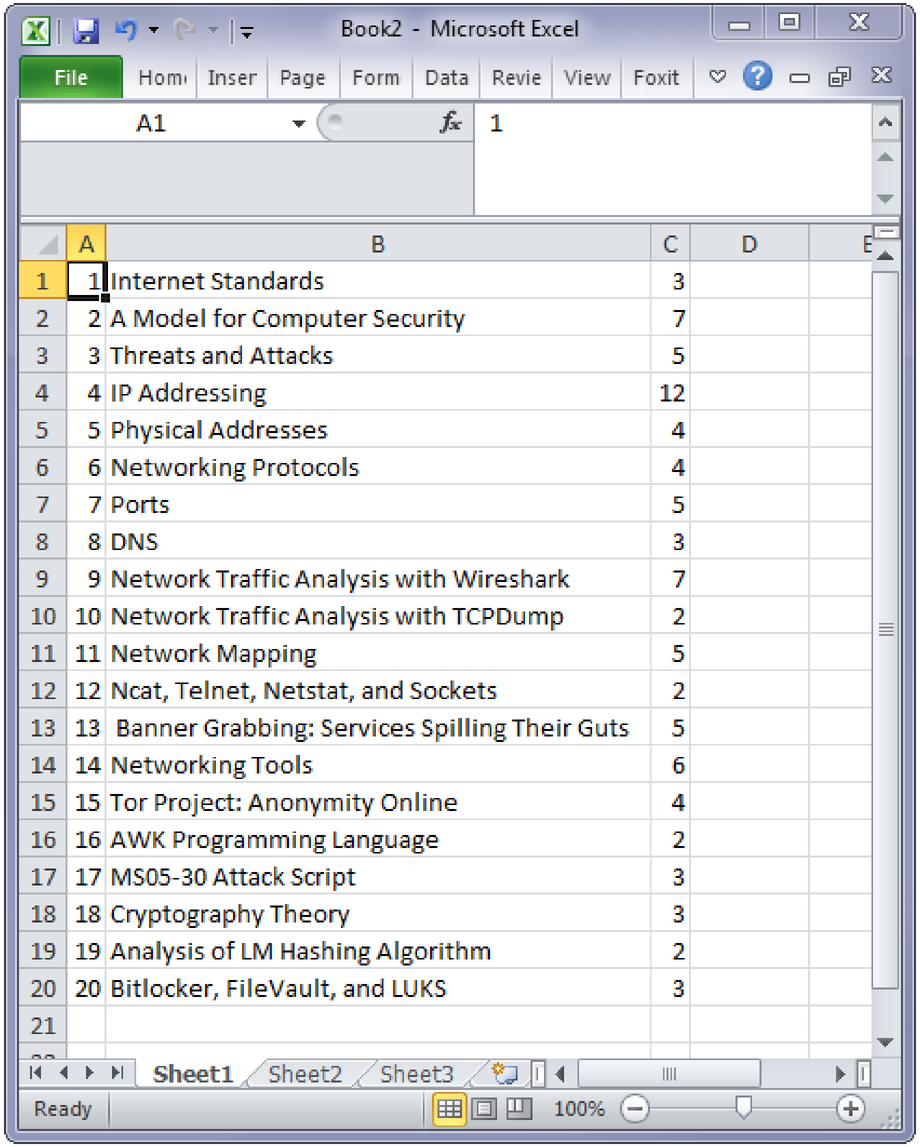
Appended >out



```
1	Internet Standards	3
2	A Model for Computer Security	7
3	Threats and Attacks	5
4	IP Addressing	12
5	Physical Addresses	4
6	Networking Protocols	4
7	Ports	5
8	DNS	3
9	Network Traffic Analysis with Wireshark	7
10	Network Traffic Analysis with TCPDump	2
11	Network Mapping	5
```

Why copy from text editor instead of shell? Shell will *render* those tabs as spaces for clipboard purposes; editor preserves them.

Planning Homework 1: Check results



	A	B	C	D
1	1	Internet Standards	3	
2	2	A Model for Computer Security	7	
3	3	Threats and Attacks	5	
4	4	IP Addressing	12	
5	5	Physical Addresses	4	
6	6	Networking Protocols	4	
7	7	Ports	5	
8	8	DNS	3	
9	9	Network Traffic Analysis with Wireshark	7	
10	10	Network Traffic Analysis with TCPDump	2	
11	11	Network Mapping	5	
12	12	Ncat, Telnet, Netstat, and Sockets	2	
13	13	Banner Grabbing: Services Spilling Their Guts	5	
14	14	Networking Tools	6	
15	15	Tor Project: Anonymity Online	4	
16	16	AWK Programming Language	2	
17	17	MS05-30 Attack Script	3	
18	18	Cryptography Theory	3	
19	19	Analysis of LM Hashing Algorithm	2	
20	20	Bitlocker, FileVault, and LUKS	3	
21				

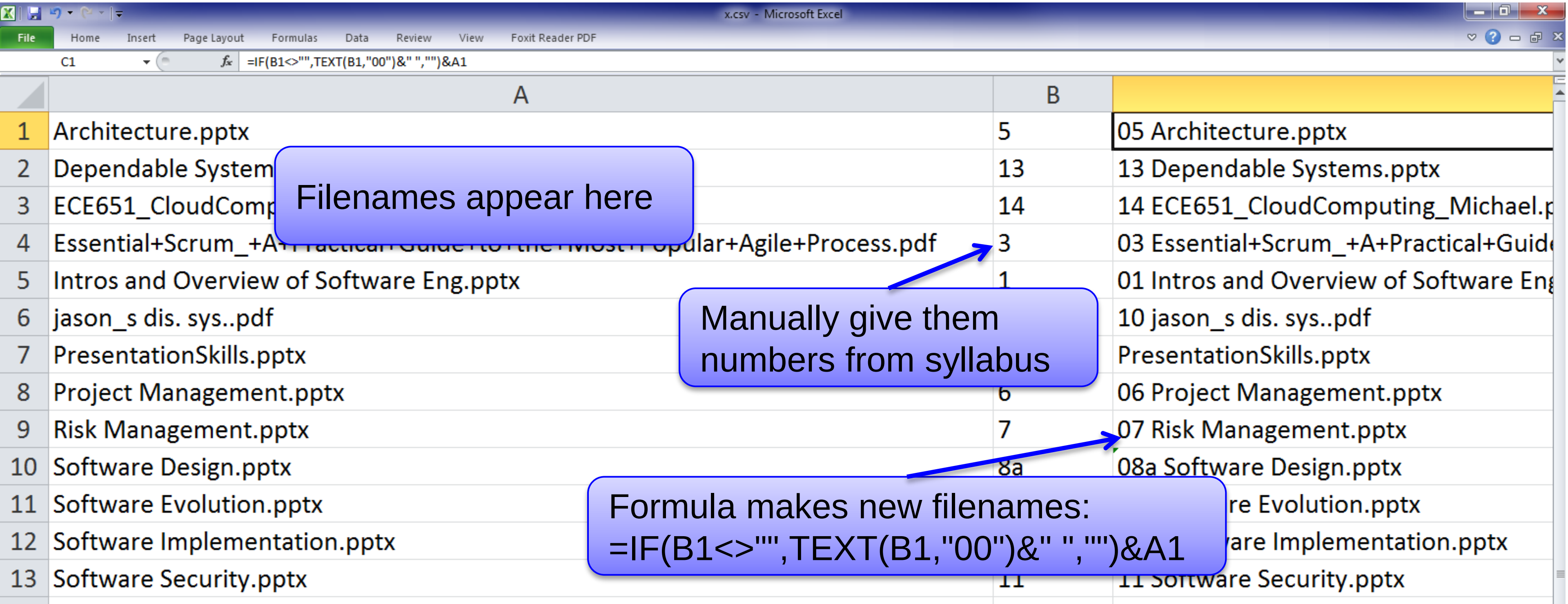
- Paste into excel
- Resize columns
- Compare a few rows against document to confirm (never forget to check that your work actually got what you think it did!)

- I can immediately see Q13 has an extra space so I fix that in the doc
- Can consider and assign points accordingly.

Example task: Organizing PPTs Gathering info

- Reviewing ECE651 course content, need to organize slides
- Have syllabus and downloaded content, want to put in order
- Can't fully automate: matching syllabus to filenames is fuzzy
- Excel can help:

```
$ ls > x.csv
```



=IF(B1<>"",TEXT(B1,"00")&" ","")&A1

	A	B	
1	Architecture.pptx	5	05 Architecture.pptx
2	Dependable System	13	13 Dependable Systems.pptx
3	ECE651_CloudComp	14	14 ECE651_CloudComputing_Michael.p
4	Essential+Scrum_+A+Practical+Guide+to+the+Most+Popular+Agile+Process.pdf	3	03 Essential+Scrum_+A+Practical+Guid
5	Intros and Overview of Software Eng.pptx	1	01 Intros and Overview of Software Eng
6	jason_s dis. sys..pdf		10 jason_s dis. sys..pdf
7	PresentationSkills.pptx		PresentationSkills.pptx
8	Project Management.pptx	6	06 Project Management.pptx
9	Risk Management.pptx	7	07 Risk Management.pptx
10	Software Design.pptx	8a	08a Software Design.pptx
11	Software Evolution.pptx		re Evolution.pptx
12	Software Implementation.pptx		are Implementation.pptx
13	Software Security.pptx	11	11 Software Security.pptx

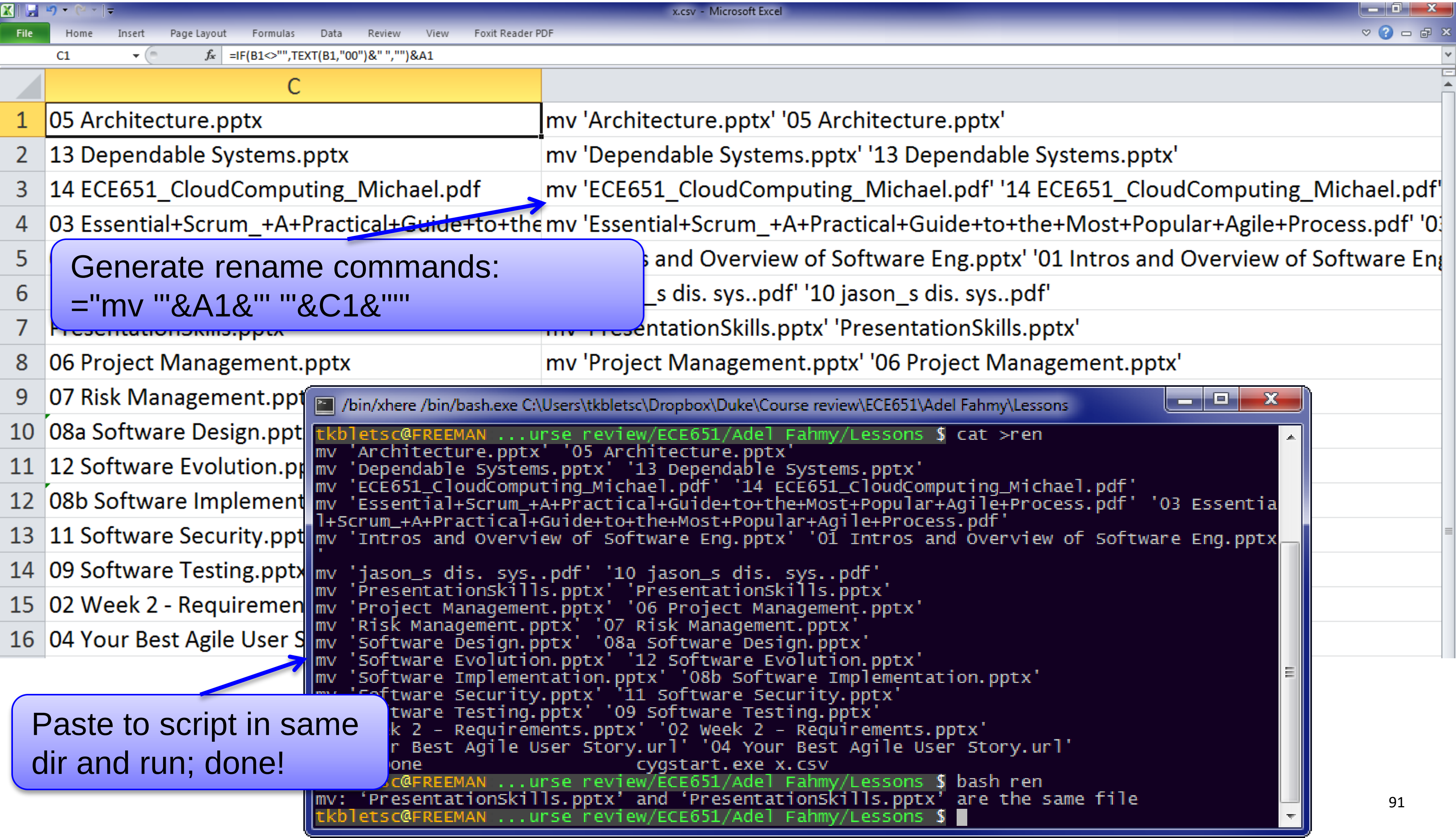
Filenames appear here

Manually give them numbers from syllabus

Formula makes new filenames:
=IF(B1<>"",TEXT(B1,"00")&" ","")&A1

Example task: Organizing PPTs Generating script and renaming

- We can even have Excel make our renaming shell script:



=IF(B1<>"",TEXT(B1,"00")&" ","")&A1

	C	
1	05 Architecture.pptx	mv 'Architecture.pptx' '05 Architecture.pptx'
2	13 Dependable Systems.pptx	mv 'Dependable Systems.pptx' '13 Dependable Systems.pptx'
3	14 ECE651_CloudComputing_Michael.pdf	mv 'ECE651_CloudComputing_Michael.pdf' '14 ECE651_CloudComputing_Michael.pdf'
4	03 Essential+Scrum_+A+Practical+Guide+to+the	mv 'Essential+Scrum_+A+Practical+Guide+to+the+Most+Popular+Agile+Process.pdf' '03
5		s and Overview of Software Eng.pptx' '01 Intros and Overview of Software Eng
6		_s dis. sys..pdf' '10 jason_s dis. sys..pdf'
7	PresentationSkills.pptx	mv 'PresentationSkills.pptx' 'PresentationSkills.pptx'
8	06 Project Management.pptx	mv 'Project Management.pptx' '06 Project Management.pptx'
9	07 Risk Management.ppt	
10	08a Software Design.ppt	
11	12 Software Evolution.pp	
12	08b Software Implement	
13	11 Software Security.ppt	
14	09 Software Testing.pptx	
15	02 Week 2 - Requiremen	
16	04 Your Best Agile User S	

Generate rename commands:
="mv '"&A1&"' '"&C1&"'"

/bin/xhere /bin/bash.exe C:\Users\tkbletsc\Dropbox\Duke\Course review\ECE651\Adel Fahmy\Lessons

```
tkbletsc@FREEMAN ...urse review/ECE651/Adel Fahmy/Lessons $ cat >ren
mv 'Architecture.pptx' '05 Architecture.pptx'
mv 'Dependable Systems.pptx' '13 Dependable Systems.pptx'
mv 'ECE651_CloudComputing_Michael.pdf' '14 ECE651_CloudComputing_Michael.pdf'
mv 'Essential+Scrum_+A+Practical+Guide+to+the+Most+Popular+Agile+Process.pdf' '03 Essentia
l+Scrum_+A+Practical+Guide+to+the+Most+Popular+Agile+Process.pdf'
mv 'Intros and Overview of Software Eng.pptx' '01 Intros and Overview of Software Eng.pptx
'
mv 'jason_s dis. sys..pdf' '10 jason_s dis. sys..pdf'
mv 'PresentationSkills.pptx' 'PresentationSkills.pptx'
mv 'Project Management.pptx' '06 Project Management.pptx'
mv 'Risk Management.pptx' '07 Risk Management.pptx'
mv 'Software Design.pptx' '08a Software Design.pptx'
mv 'Software Evolution.pptx' '12 Software Evolution.pptx'
mv 'Software Implementation.pptx' '08b Software Implementation.pptx'
mv 'Software Security.pptx' '11 Software Security.pptx'
   tware Testing.pptx' '09 Software Testing.pptx'
   k 2 - Requirements.pptx' '02 Week 2 - Requirements.pptx'
   r Best Agile User Story.url' '04 Your Best Agile User Story.url'
   one                     cygstart.exe x.csv
   c@FREEMAN ...urse review/ECE651/Adel Fahmy/Lessons $ bash ren
mv: 'PresentationSkills.pptx' and 'PresentationSkills.pptx' are the same file
tkbletsc@FREEMAN ...urse review/ECE651/Adel Fahmy/Lessons $
```

Paste to script in same dir and run; done!

Conclusion

- Time it took to do this: 3 minutes
- Time it would have taken to do this manually: 6 minutes?
  - Odds of a typo or transcription error: way higher than automated way
- Time it would take to do this if I were learning the skills for the first time: way more than 6 minutes.

- So is it worth it to learn to automate?
  - **Yes: you learn once, benefit *many* times. ☺**
  - One of the sources of developer or sysadmin productivity!
- Keep doing things the dumb manual way because it’s faster?
  - You never improve! ☹

Conclusion: PRACTICE THIS STUFF!!