Shell First Steps

5 minute read

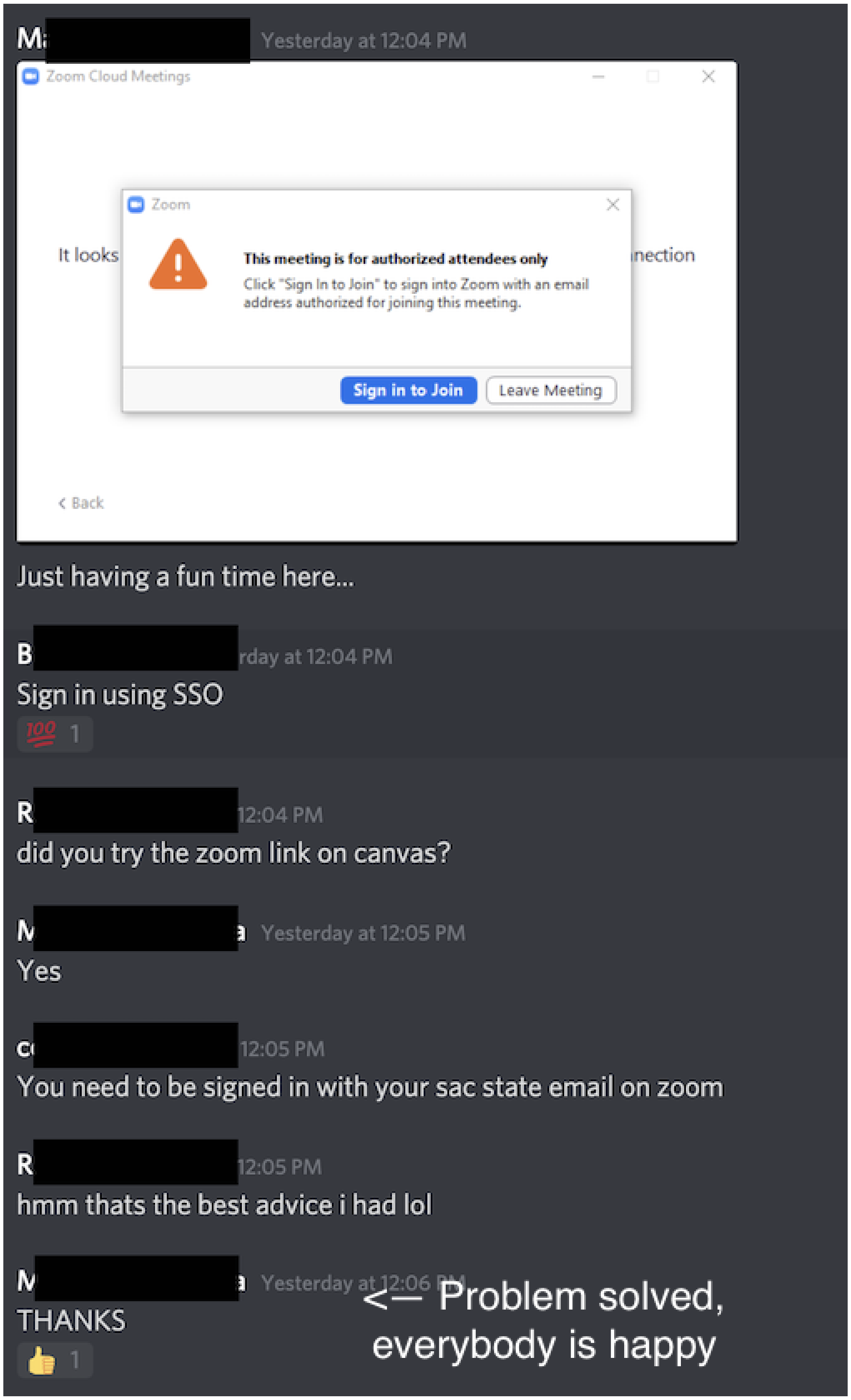
- Explain reasons to use the shell
- Do basic operations on files and directories, such as listing, creating, deleting, moving, and copying

Announcements

- Our class Discord is an extension of our classroom. It is an inclusive, welcoming place.

123 GO: Come up with one example of either good or bad online classroom behavior. By bad, we mean anything that takes away from student's learning, or could make others feel unwelcome. For example:

- Bad: Sending private messages to someone who's not interested.
- Good: Helping someone through a technical issue.



M [redacted] Yesterday at 12:04 PM

Zoom Cloud Meetings

It looks [...] nection

Zoom

This meeting is for authorized attendees only

Click "Sign In to Join" to sign into Zoom with an email address authorized for joining this meeting.

Just having a fun time here...

B [redacted] rday at 12:04 PM

Sign in using SSO

💯 1

R [redacted] 12:04 PM

did you try the zoom link on canvas?

M [redacted] a Yesterday at 12:05 PM

Yes

c [redacted] 12:05 PM

You need to be signed in with your sac state email on zoom

R [redacted] 12:05 PM

hmm thats the best advice i had lol

M [redacted] a Yesterday at 12:06 PM

THANKS

👍 1

<— Problem solved, everybody is happy

Resources

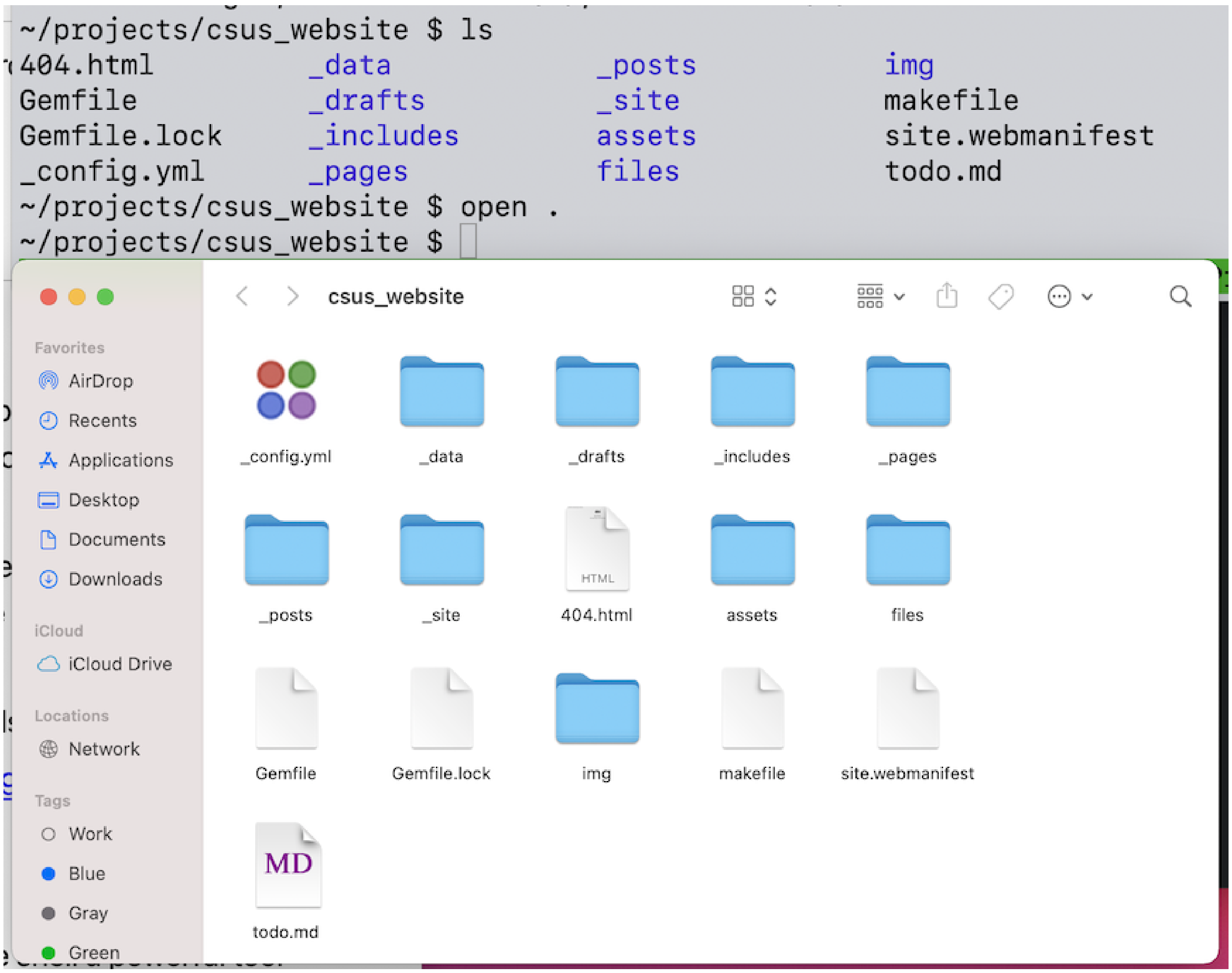
- Software Carpentry Unix Shell Lesson (https://swcarpentry.github.io/shell-novice/)
- notes from a previous lecture on shell (https://github.com/clarkfitzg/sta141c-winter19/blob/master/lecture/01-31-bash1.md#shell)

What's the shell?

The shell is a text-based way to interact with your computer and run commands. Graphical User Interface's (GUI's), in contrast, have been far more popular since at least the 1990's.

The shell is also known as: terminal, command line, command line interpreter, UNIX shell, POSIX shell, command prompt, console. It comes in several flavors: bash, zsh, ksh, tcsh, fish. Windows also has PowerShell and others, which are the same concept, but different syntax, and we won't use it for this class.

Here we can see a shell and an open file explorer both listing the contents of my `csus_website` directory.



```
~/projects/csus_website $ ls
404.html         _data            _posts           img
Gemfile          _drafts          _site            makefile
Gemfile.lock     _includes        assets           site.webmanifest
_config.yml      _pages           files            todo.md
~/projects/csus_website $ open .
~/projects/csus_website $ 
```

Motivation

Reasons to use the shell:

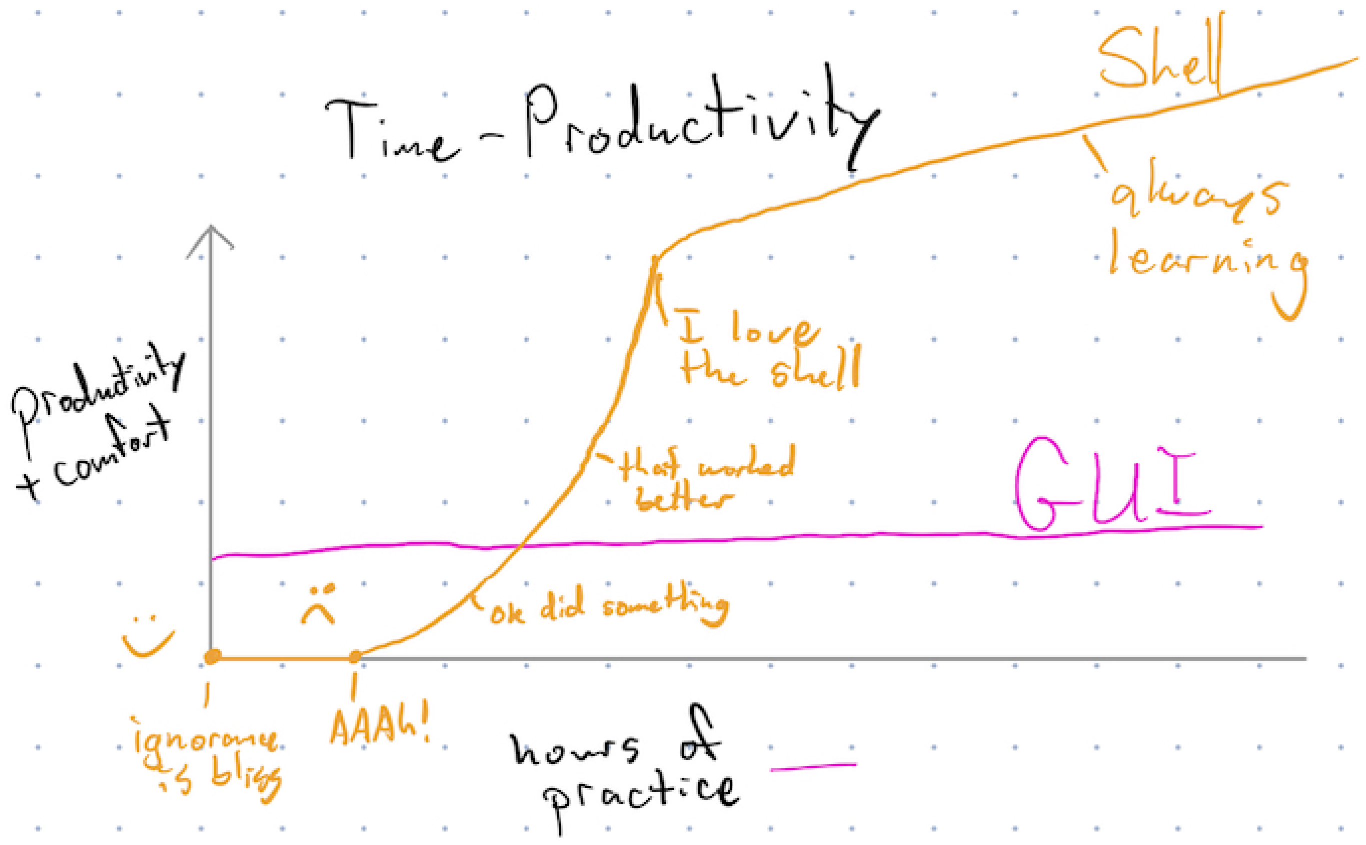
1. The shell can interact with remote machines. GUI’s aren’t always available; the shell is. For us, the shell will be the **only** way to interact with our machines. By using it locally, we get relevant experience.
2. The shell is more efficient and precise than GUI’s. For example, suppose you have 100 `.csv` files mixed in the same directory with 1000 other files, and you want to move only the `.csv` files to another directory. This is trivial with the shell.
3. We can easily automate commands in the shell by saving the commands in a file.
4. The shell is stable. For example, `ls` has been listing directories since 1971 (https://linuxgazette.net/issue48/fischer.html). Learn these tools, and they’ll serve you for your whole career.
5. Many programs have only a shell interface, for example, `aws s3`.
6. The shell can compose programs with `|`, the pipe. This idea makes the shell a powerful tool for processing raw text data, and we’ll practice this technique in this class.
7. The shell has much in common with other programs, particularly those with roots in UNIX. When you use them all, there is a synergistic effect in your efficiency. For example, what is the function to list objects in an environment in R? `ls()`, from the UNIX `ls`. There’s also `rm`, `head`, `grep`, … the list goes on :) The magic commands in IPython (https://ipython.readthedocs.io/en/stable/interactive/magics.html) are similar.
8. Compared to GUI’s, the shell makes it easier to describe and share a sequence of commands with others, for example, to install software.
9. The shell can be customized to your heart’s content.

Reasons NOT to use the shell:

1. There’s a learning curve.
2. Commands can be cryptic.

Time Productivity Curve

Get over the initial period of frustration, and life will be great.



Don’t Type the Leading $

The `$` preceding each command denotes the prompt where the user can interactively type commands. The preceding `$` usually refers to a complete, syntactically valid command, rather than just the name of a command. For example `$ mv a.txt b.txt` is a complete command, while `mv` by itself is the name of a command. Interactively, we’ll see something like the following:

```
~ $ echo "a" >> a.txt
~ $ mv a.txt b.txt
~ $
```

The final line, `~ $`, shows the shell is ready for another command.

Essential Concepts

- The **working directory** is the location of the shell in the file system, analagous to the directory you open in a GUI file explorer.
- Two ways to identify files and directories:
  - An **absolute path** unambiguously identifies files and directories within the file system. Absolute paths start with `\`, the root of the directory tree.
  - A **relative path** gives the location of files and directories relative to the working directory.
- `~` means home directory, so `$ cd ~` means changes the working directory to your home directory.
- `..` means the parent directory, so if my working directory is `/Users/fitzgerald/projects`, and then I run `$ cd ..`, then my working directory will be `/Users/fitzgerald`.
- `*` is a **wildcard** that matches any character. It can be combined with most of the commands below to operate on many files simultaneousy. For example, `$ rm *.txt` removes all files that end in `.txt` within the working directory.
- Arguments appear following the commands preceded by `-` or `--`. For example, `$ ls -R` lists all directories recursively from the working directory. The `-R` is for recursive.

Shortcuts

- Press tab to autocomplete commands and file names. For example, suppose I have a file called `2021-01-26-shell-first-steps.md` and I want to type the command `$ cat 2021-01-26-shell-first-steps.md`. If there are no other files that begin with `2021-01-26`, then I can type `$ cat 2021-01-26`, press tab, and have my command.
- Cycle through previous commands by pressing the up arrow.

Commands

🧨 below denotes a command that can delete your files. Proceed with caution, because you CANNOT undo.

Command	Mnemonic	Example
`cat`	concatenate	`$ cat README` prints out the contents of the file `README`
`cd`	change directory	`$ cd bicycles` sets the working directory to `bicycles`
`clear`	clear	`$ clear` clears the output of the previous commands
`cp` ✏	copy	`$ cp a.txt b.txt` copies `a.txt` to `b.txt`, and deletes the old `b.txt`.
`echo`	echo text	`$ echo hello` prints out `hello`. `$ echo hello >> README` appends the line `hello` to the file `README`.
`ls`	list directory	`$ ls` lists the contents of the working directory.
`mkdir`	make directory	`$ mkdir bicycles` creates a new directory called `bicycles` within the working directory
`mv` ✏	move	`$ mv a.txt b.txt` renames `a.txt` to `b.txt`, and deletes the old `b.txt`.
`pwd`	print working directory	If `$ pwd` displays `/Users/fitzgerald/projects/csus_website`, then this is the absolute path to my working directory.
`rm` ✏	remove	`$ rm b.txt` deletes the file `b.txt`.

Exercise

Open up a shell and perform the following tasks. Turn in the commands you used, together with output.

1. Create a new directory `stat196k_exercise1` in your home directory.
2. Change your working directory to `stat196k_exercise1`.
3. Check the absolute pathname of the current working directory.
4. Create two files `data1.csv` containing just a single `1`, and `data2.csv` containing just a single `2`.
5. Verify the contents of these two files.
6. Create a directory called `data`.
7. Move all the `.csv` files into the directory `data` using a single command. This same command should work for 2 files, or 2,000 files.
8. Verify the files are in the directory `data`, and not in `stat196k_exercise1`.
9. Copy the `.csv` files from `data` into `stat196k_exercise1`.
10. Verify using a single command that you have two copies of each `.csv` file, one in `stat196k_exercise1` and one in `stat196k_exercise1/data`.
11. Remove the `stat196k_exercise1` directory, and everything in it.

Updated: January 26, 2021