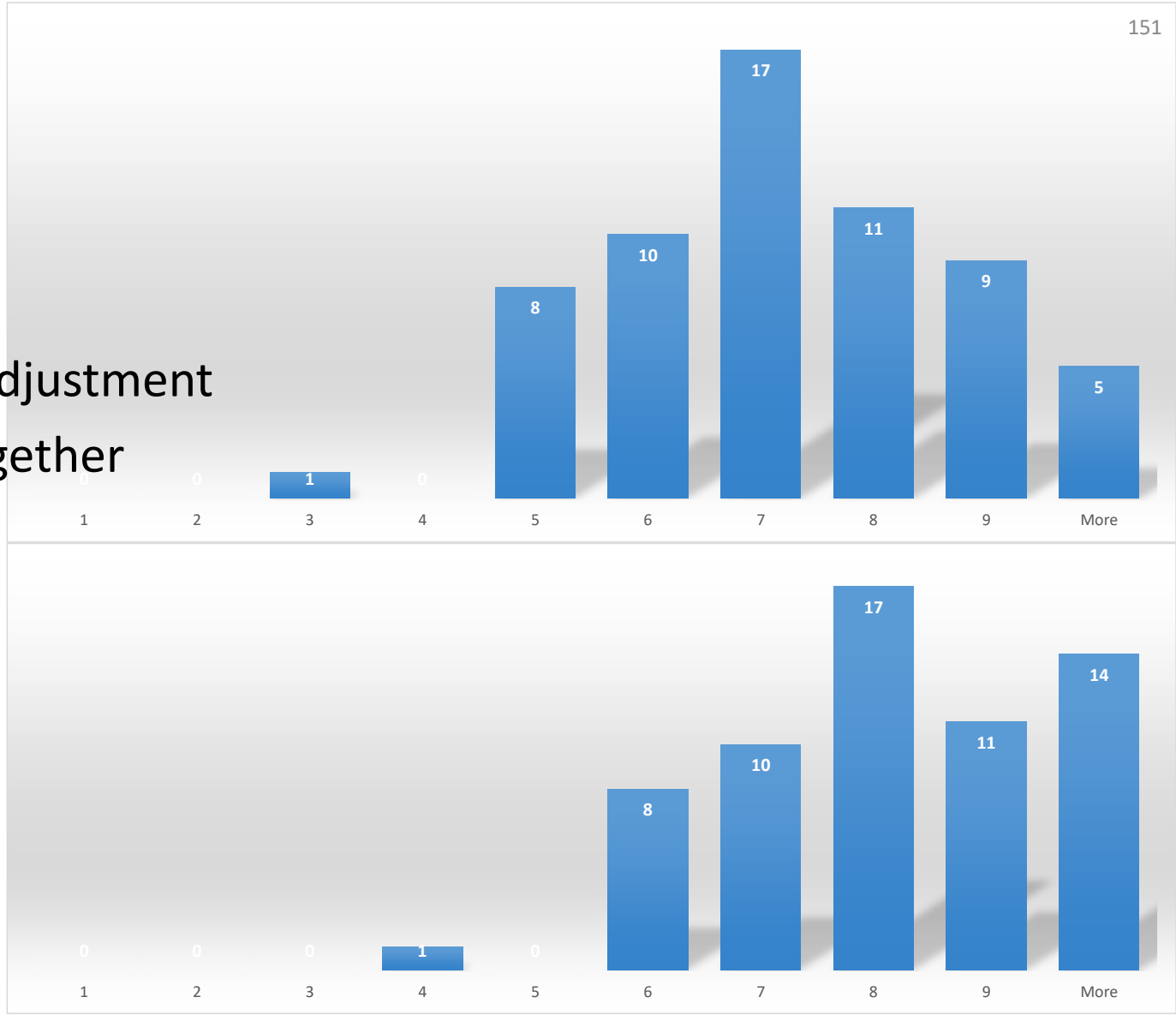
Quick Announcements

- IE0 done being graded by our TAs
- Still some unfinished business with M2
- M3 will not follow COP3353 but jump ahead to around M6 (for those of you following on the playlist)

IEO

- Done grading by TAs
- Tad too long, needs adjustment
- + review solutions together



Useful Commands

`env` command

- → a variable like `value=99` is not listed
- Note that lots of variables are defined already, more about this later

`unset` command

- `SOMETHING=42`
- `echo $SOMETHING`
- `unset SOMETHING`
- `echo $SOMETHING`

Predefined Vars in the Bash Environment

- Some vars are predefined, we can add our own
- Here is an overview of some of the default vars available in the bash environment:

- `echo $HOME`
- `echo $USER` → these depends on who logged in & started shell
- `echo $SHELL`
- `echo $HOSTNAME` → in other shells; `echo $HOST`
- `echo $PWD` → same as `pwd` command

https://youtu.be/X8ANOYOT5mc

Let's talk about The PATH

`echo $PATH`

- Look at folders; they all relate to BINaries / executable programs
- Role = folders in which we look up for external commands

`whereis ls`

How to append to the PATH?

Let's talk about The PATH

```
echo $PATH
```

- Look at folders; they all relate to BINaries / executable programs
- Role = folders in which we look up for external commands

```
whereis ls
```

How to append to the PATH?

- @Beginning `export PATH=$PATH:/home/tux/project/bin`
- @End `export PATH=/home/tux/project/bin:$PATH`
- Convention: append our local stuff at the end

RANDOM

`echo $RANDOM`

- It holds a dynamically generated value
- Range = ????

How do we figure out the range

RANDOM

`echo $RANDOM`

- It holds a dynamically generated value
- Range = ????

How do we figure out the range

→ Refer To the Fantastic Manpage for bash:

RANDOM *Each time this parameter is referenced, a random integer between 0 and 32,767 is generated. The sequence of random numbers may be initialized by assigning a value to RANDOM. If RANDOM is unset, it loses its special properties, even if it is subsequently reset.*

Modifying the shell prompt

It can be modified via env variable `PS1` and a special syntax

```
echo $PS1
```

- Using escape sequences to colorize things
- Escape codes that are replaced by hostname or the likes e.g., \w
  - \u username
  - \w current working directory
  - \h hostname
- ${ } → advanced syntax substituting a value if var is not defined

```
export PS1="What now? "
```

PS1 Syntax

See man bash

Symbol	Role
\w	Current working directory, using ~ shortcut notation if applicable
\W	Basename of the current working directory
\h	Hostname
\u	Username
\s	Name of the shell
\@	Time of the day (am/pm)
\A	Time of the day (24hrs)
\d	Date
\!	Event number in the history list (skip)

Environment Variables to customize PS1

Variable	Role
HOME	Location of the user's home directory
HOSTNAME	Hostname of the computer
HOSTTYPE	Architecture of the computer (e.g., arm64)
LOGNAME	username
PWD	Current working directory
RANDOM	Random number
SECONDS	Time from start of current shell (in seconds)
SHELL	Path to shell's executable
USER	username

Exercise: I want a lucky number!

- We want a PS1 prompt that says, "Your lucky number is: 1234, what now? ”
- Now that it works with 1234, we want to instead use the RANDOM variable

Provide the necessary
command line to achieve this

It's true! RNG is not random!

`export PS1="Your lucky number is ${RANDOM} "`

- Wait! The random number is the same every time
- Bc we used " " `$RANDOM` was replaced by a random number
- The `PS1` was evaluated one time then set to whatever `$RANDOM` returned

What if I want a prompt that shows a new random number each time?

- `export PS1='Your lucky number is ${RANDOM}'`
- Single quote means that `PS1` is set to a string containing `${RANDOM}`
- So each time it is used, it will get evaluated to a new random number

Little Experiment….

```
export PS1='$RANDOM'
echo $PS1
```

→ PS1 is a string containing $RANDOM itself

```
export PS1="$RANDOM"
echo $PS1
```

→ it's a string with just a random number in it, always the same

Are other variables updated the same way?

Can you think of other BASH predefined variables that might work the same way when integrated in a prompt?

Are other variables updated the same way?

Can you think of other BASH predefined variables that might work the same way when integrated in a prompt?

- Would work the same with `$PWD $SECONDS` since they are also dynamically generated

- `$USER $HOSTNAME` might be generated only when starting the bash

Let’s go back to listing all env variables!

- `env` or `printenv` to check all env variables

- `env | less`
  - We saw less already
  - We have not talked about piping yet

https://youtu.be/Ej3I9i5feOI

How do we differentiate env vars vs. shell var?

- `MYVAR="this is a global / environment variable"`
- `export MYVAR`
- `myvar="this is a local / shell variable"`
- `echo $myvar`
- `echo $MYVAR`

`env` shows `MYVAR` but not `myvar`

Trick to not have to read through the whole listing

- `env | grep "MYVAR"`
- `env | grep "myvar"`

Is there another way to list variables?

`set` command shows:

- env variables (like env / printenv)
- but also local variables,
- then functions defined in the shell

Common Usages

- `set | less`
- `set | grep "MYVAR"`
- `set | grep "myvar"`

`Set` gives info about both local / global vars

2nd use of set command: list & (un)set options

- Env variables can be used to configure the shell (e.g., PS1)
- We also have special variables called options

- `set -o` lists all available options & if they are on/off
- `set -o option_name` turn option ON
- `set +o option_name` turn option OFF
- `set +o` see next slide…

Using set +o

`set +o`

- Unlike in previous slide, no name given
- Displays list of options but tells us whether they are on or off in the form of the command that was used on that option

Illustrating the difference:

- `set -o | grep ignoreeof` → shows that it is off
- `set +o | grep ignoreeof` → shows it has been set to +o

What could it possibly be used for?

- Dump in a file the list of commands to set the same options in another shell
- E.g., put that in a bash init file (next topic)

Example: `ignoreeof`

Focus on one as illustrative example: `Ignoreeof`

- Off by default
- ^D shortcut → show what it does on new terminal
  - Generates EOF special char
  - When bash sees that, it considers input is over and therefore exits

We can change that default behavior

- `set -o ignoreof` → turn ON the option
- `set -o` → look to verify option is on
- Test what happens when we hit ^D → msg displayed but shell does not exit

- `set +o ignoreeof` → Now the option is OFF

Topic

M2T3
Bash Initialization Files

https://youtu.be/DDYw1b29wk8

Before we get started….

Whose shell are we running?

→ you can have multiple bash shells running for any given user

Where are the configuration files located?

- `~/` → specific to a given user
- `/etc/` → affect all users

What kind of shell are we running?

- Interactive vs. non-interactive bash shells
- Login s. non-login bash shell

How to identify the different types of shells

Type of Shell	How to identify it
Interactive	You can type commands
Non-Interactive	You cannot (You read commands from a file (script))
Login	You had to provide credentials when starting the shell
Non-Login	You did not

How to start the different types of shells

Type of Shell	Interactive	Non-Interactive
Login	• `CTRL + ALT + F3` to virtual console (then `CTRL + ALT + F2` to go back to the GUI) • `Bash --login` does not prompt credentials but follows all other steps • `ssh alessio@penguin.edu`	• `echo "ls -l ~/" | ssh alessio@penguin.edu` • `bash --login -c 'ls -l /'`
Non-Login	• Type bash in another shell • Open new terminal window • Open new tab in terminal window	• `bash myscript.sh` • `./myscript.sh`

FROM → https://askubuntu.com/questions/879364/differentiate-interactive-login-and-non-interactive-non-login-shell

How to tell if a shell is interactive?

- Check the contents of the $- variable.
  - For interactive shells, it will include i
- Interactive shell, just running a command in a terminal
  - `echo $- himBHs`
- Non-interactive shell
  - `bash -c 'echo $-' hBc`

FROM → https://askubuntu.com/questions/879364/differentiate-interactive-login-and-non-interactive-non-login-shell
More About $- at https://stackoverflow.com/questions/42757236/what-does-mean-in-bash

How to tell if the shell is a login shell?

- There is no portable way of checking this but, for bash, you can check if the `login_shell` option is set:
- Normal shell, just running a command in a terminal: interactive
  - `shopt login_shell`

    `login_shell off`
- Login shell;
  - `ssh localhost`

    or
  - `bash --login`
  - `shopt login_shell`

    `login_shell on`

FROM → https://askubuntu.com/questions/879364/differentiate-interactive-login-and-non-interactive-non-login-shell

Putting it all together….

Type of Shell	Interactive	Non-Interactive

Login	`bash -l`
	`echo $-`
	`himBHs`
	`shopt login_shell`
	`login_shell on`
		`echo 'echo $-;shopt login_shell'|ssh 127.0.0.1`
		(Pseudo-terminal not allocated since stdin is not a terminal.)
		`hBs`
		`login_shell on`

Non-Login	`echo $-`
	`himBHs`
	`shopt login_shell`
	`login_shell off`
		`bash -c 'echo $-; shopt login_shell'`
		`hBc`
		`login_shell off`

Back to looking up initialization files in the Bash manpage

- Focus on `INVOCATION` section
- Let's clarify it a bit what the manpage says:

Type of Shell	Interactive

Login	1. `/etc/profile`
	2. 1st of the following that exists:
	`~/.bash_profile`
	`~/.bash_login`
	`~/.profile`

Non-Login	1. `/etc/bash.bashrc` 2. `~/.bashrc`

A little experiment

```
sudo gedit /etc/profile /etc/bash.bashrc ~/.profile
~/.bashrc &
```

- Add `echo` commands to each of them, start with `/etc/profile`

- Enter bash to a bash prompt → not triggered
- `bash --login` → triggers it

…and so on so forth…

What the manpage says… vs our experiment

Type of Shell	Interactive
Login	1. /etc/profile 2. /etc/bash.bashrc 3. 1st of the following that exists: ~/.bash_profile ~/.bash_login ~/.profile 4. ~/.bashrc
Non-Login	1. /etc/bash.bashrc 2. ~/.bashrc

Why are the yellow highlights also loaded?

What the manpage says… vs our experiment

Type of Shell	Interactive
Login	1. /etc/profile 2. /etc/bash.bashrc 3. 1st of the following that exists: ~/.bash_profile ~/.bash_login ~/.profile 4. ~/.bashrc
Non-Login	1. /etc/bash.bashrc 2. ~/.bashrc

Why are the yellow highlights also loaded?

/etc/profile calls
. /etc/bash.bashrc

Hard to introduce configurations that affect only non-login shells, as long as the profile files can call the local ones too

Remember that users can do what they want in their ~/.profile e.g., loading ~/.bashrc or even /etc/bash.bashrc