First Introduction to Linux

Prof. Adam Teman

EnICS Labs, Faculty of Engineering

Bar-Ilan University

5 February 2024

Emerging Nanoscaled Integrated Circuits and Systems Labs

The Alexander Kofkin Faculty of Engineering

Bar-Ilan University

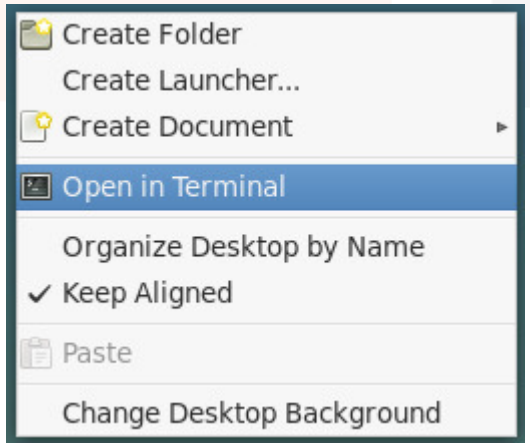
Linux

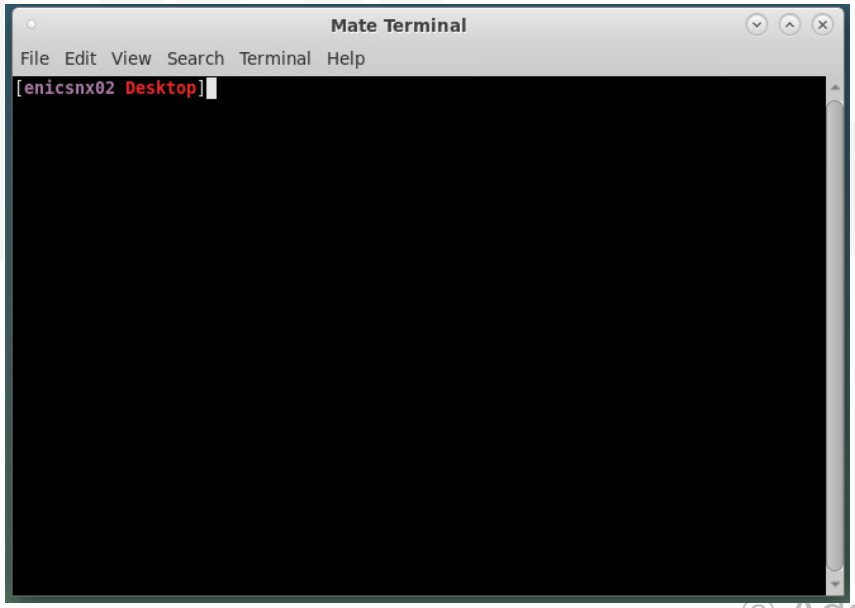
- **Linux is a family of open-source Unix-like operating systems.**
  - The Linux kernel was released by Linus Torvalds in 1991.
  - Provided under the GNU General Public License.
- **Originally developed to provide a Unix experience for personal computers based on x86**
  - Currently ported to more platforms than any other OS.
  - Android is based on Linux.
- **Linux is usually packaged as a distribution or "Distro"**
  - Red Hat, Fedora, Ubunto, CentOS, SUSE, others
- **Commonly distributed with windowing system and desktop environment (e.g., GNOME, KDE)**

Photograph by Kimmo Mantyla

The Bash Shell

- **Your interface into the operating system is the "shell"**
  - Allows you to run programs
  - Give input to programs
  - Inspect the output of programs
- **The "Bourne Again Shell" (bash) is the most popular Linux shell today.**
  - We will first open a "terminal".
  - This will provide us with a "prompt"





Hello, World!

- **Every programming course starts with a "Hello, World!"**
  - To tell bash to print "Hello, World!", we'll use the command `echo`:

```
$ echo 'Hello, World!'
```

  - `echo` is the name of the program and `'Hello, World!'` is the argument.

- **We can run other programs, try, for example, `date`:**

```
$ date
```

Paths

- **How does the shell know how to find the `date` or `echo` programs?**
  - It searches through a list of locations on the file server.
- **Where is this list stored?**
  - In an *environment variable* called `PATH`.
- **To dereference a variable, we will use the `$` character:**

```
$ echo $PATH
```

  - We got a list of locations on the server, which are used to search for programs.
- **But where did it find the `echo` and `date` programs?**

```
$ which date
```

  - We see that these are executable files stored in the `/bin` (=binaries) folder
  - Alternatively, we could have run:

```
$ /bin/echo 'Hello, World!'
```

Navigating in the Shell

- **So we saw that there are "locations" in the Linux environment**
  - `/` is the "root" of the filesystem, under which all directories and files lie.
  - `~` is your "home" directory, but this is an alias.
  - To see what the real path to your home directory is:

```
$ pwd
```

```
$ echo $HOME
```

  - `pwd` is short for "print working directory" – that's "where we are" now
- **We can navigate through directories with `cd` (change directory)**

```
$ cd ..
```

```
$ cd ./temanad
```

  - `.` is the "current" directory, while `..` is the "parent" directory

Directory Contents

- To see what files and directories are in the current folder, use the `ls` command

```
$ ls
```

```
$ ls ./Downloads
```

- Add flags and options (usually starting with a `-`) to modify a command's behavior

```
$ ls -l
```

- To get a list of options use the `-h` or `--help` flag or open the `man` page

```
$ ls --help
```

```
$ man ls
```

- Use "globbing" to match many strings
  - `?` Matches any single character

```
$ ls myfile?
```

  - `*` matches any one or more characters

```
$ ls myfile*
```

Dot Files

* because the file names begin with a `.`

- **Many programs are configured using plain-text files known as dotfiles***
  - Filenames that start with a `.` are hidden by `ls` unless the `-a` flag is used.

```
$ ls -a
```

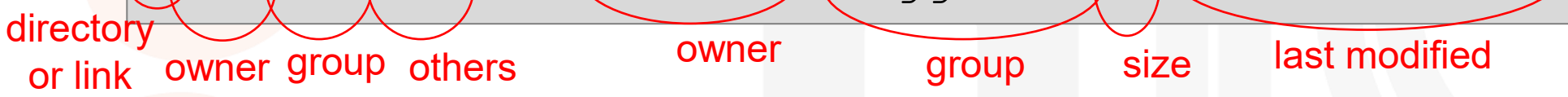
- **Some important dotfiles are:**
  - `~/.bashrc`, `~/.bash_profile`: Configure settings for your Bash shell.
  - `~/.gitconfig`: Configure git.
  - `~/.vimrc`: Configure VIM.
  - `~/.ssh/config`: Configures secure shell (ssh).

File Permissions

- **Files are created with default permissions (read/write/execute access)**
  - Create an empty file with the `touch` command
  - Show information about the file with `ls -l`:

```
$ touch myfile
```

```
$ ls -l myfile
-rw-r--r--. 1 temanad engguest 0 Mar 23 12:36 myfile
```



- **To change file permissions, use the `chmod` command:**
  - Make the file executable: `chmod +x myfile`
  - Make the file writeable by other group members: `chmod g+w myfile`
  - Use a bit mask to make the file readable/writeable by all: `chmod 666 myfile`

Redirection

- **By default the input/output of your program is the terminal:**
  - Input is from your keyboard
  - Output is to the screen
- **But you can "redirect" the input/output streams using `< file` and `> file`:**
  - Print "hello" to a file instead of the screen

```
$ echo "hello" > myfile
```

  - To see that it worked, use the `cat` command:

```
$ cat myfile
```

  - Now *redirect* our file to be used as the input to the `cat` command and write the output into a new file:

```
$ cat < myfile > myfile2
```

  - Append `"world!"` to the file

```
$ echo `world!' >> myfile2
```

Other basic commands

- **Create a directory**

```
$ mkdir mydir
```

- **Remove a directory**

```
$ rmdir mydir
```

- **Copy a file**

```
$ cp myfile myfilecopy
```

- **Rename (move) a file**

```
$ mv myfilecopy myfile2
```

- **Delete a file**

```
$ rm myfile2
```

- **Finding a file**

```
$ find . -name myfile
$ find . -name myfile -exec rm {} \;
```

Other basic commands

- **Seeing command history**

```
$ history
```

- **Viewing files**

```
$ cat myfile
$ more myfile
$ less myfile
$ vim myfile
$ nano myfile
```

- **Show the beginning or end of a file**

```
$ head myfile
$ tail myfile
```

- **Compare files**

```
$ diff myfile myfile2
```

Pipes

* Usually find this character using `shift+\` next to the `return` key on your keyboard.

- **The pipe (|) operator* lets you "chain" programs such the output of one is the input of another:**

```
$ ls -l | grep my*
```

  - In this example, we took the output of the `ls -l` command and sent it to the `grep` command.
  - `grep` is an extremely powerful shell command that lets you select lines of text in a file that match a given string. In this case, if the line of text contains any word starting with "my" (e.g., `myfile`, `myfile2`) then they will be printed out.

- **You can get the output of a command as a variable using `$(CMD)`**

```
$ echo "The current date is $(date)"
```

Aliases and Symbolic Links

- Instead of writing out a whole (complex) command, use an alias with the syntax

```
alias alias_name="command_to_alias arg1 arg2"
```

```
$ alias ll="ls -ltrh"
$ alias gv="grep -v"
$ alias grl="grep --color --line-number''
```

- To see a list of all configured aliases, type `alias`.

```
$ alias | less
```

- And you can create a link (shortcut) to a file or directory for quick access:

```
$ ln -s myfile mylink
```

Writing an executable program

- **Let's start by writing an executable "Hello, World!" program:**
  - Create a file that prints out "Hello, World!":

```
$ echo 'echo "Hello, World"' > hello
```

* Note our use of single and double quotes.

- **Now try to execute the file:**

```
$ ./hello
```

  - Ah, we need to make it executable… `chmod u+x hello`
- **But how can we tell it to use Bash (and not something else) to run our program?**
  - We'll use the very popular `VIM` text editor:

```
$ vim hello
```

  - Press "`i`" to go into "insert" mode.
  - Now type:

```
#!/bin/bash
echo "hello"
```

  - Hit `esc` to exit "insert" mode.
  - To save and exit, type `:wq`

Variables

- **The shell, like other programming languages, has variables.**
- **In Bash, we just write `var=value` (no spaces!) to define a variable**

```
$ foo=bar
```

  - Pay attention that using quotations (“”) will substitute values, while ‘’ will not:

```
$ echo “$foo”
```

```
$ echo ‘$foo’
```

- **Variables are local to the shell, so they aren’t known to programs**
  - Instead, you can use *environment variables*, such as `$PATH`, `$HOME`.
  - You can access environment variables from within programs.
  - To see a list of environment variables, type `env`.
  - To define a new environment variable, type `export`:

```
$ env | more
```

```
$ export charlie=brown
```

Shell Scripting

- **Bash supports regular control flow commands, such as `if`, `case`, `while`, `for`.**
- **In addition, you can write scripts, and pass arguments to them:**
  - `$0` – name of the script
  - `$1`-`$9` – arguments
  - `$@` all the arguments
  - `$#` number of arguments
  - `$?` – exit status of previous

```
#!/bin/bash

echo "Running program $0 with $# arguments"
for file in "$@"; do
  grep foobar "$file" > /dev/null 2> /dev/null
  # If pattern not found, grep has exit status 1
  # Redirect STDOUT and STDERR to a null register
  if [[ $? -ne 0 ]]; then
      # If grep exited with status 1
      echo "Adding foobar to $file"
      echo "# foobar" >> "$file"
  fi
done
```

- **You can also call a script written in another language:**

```
#!/bin/python
print ("hello, world!")
```

Compressing and Uncompressing

- To compress a file in Linux, you can use the `zip` command:

```
$ zip myfile.zip myfile.txt
```

- To uncompress a file:

```
$ unzip myfile.zip
```

- But in Linux we often compress a whole folder using `tar`:

```
$ tar -czvf name-of-archive.tar.gz /path/to/directory-or-file
```

- Then extract the archive:

```
$ tar -xzvf archive.tar.gz
```

Job Control

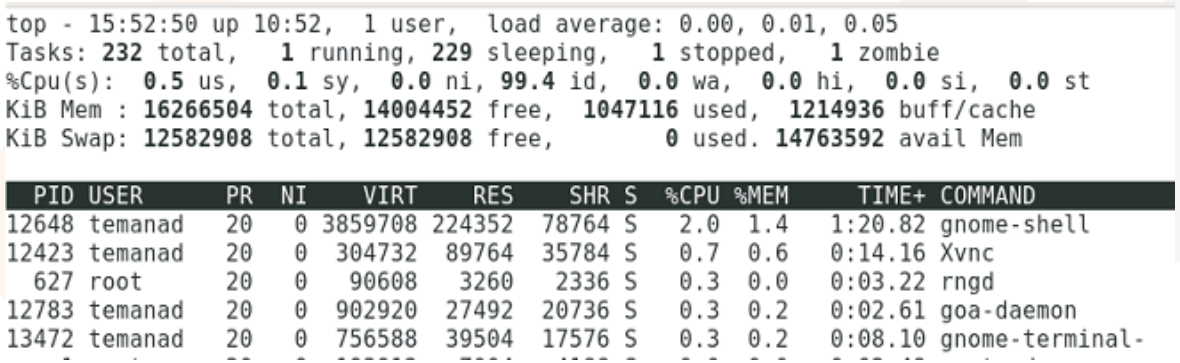
- **Sometimes you need to send a *software interrupt* to your process, while it is still running (or possibly stuck…)**
  - `Ctrl-c`: Sends a `SIGINT` signal to the process, usually killing it.
  - `Ctrl-\`: Sends a `SIGQUIT` signal to the process, killing it.
  - `Ctrl-z`: Sends a `SIGSTP` signal that pauses a process.
    To continue a process after pausing it, type the `fg` command.
    To continue running the process in the background, type the `bg` command.
  - To start a command running in the background, use `&`.

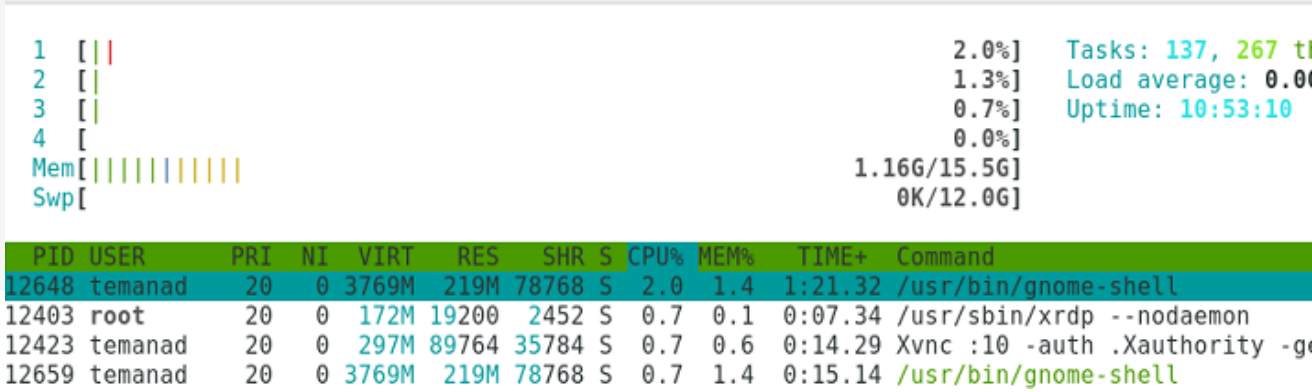
```
$ firefox &
```

- **To see all unfinished jobs (run from this terminal) type `jobs`.**
  - The jobs are listed as `[n]`. You can control the specific job with `%n`.
  - For example, move the first process to the foreground using `fg %1`.
  - Kill the second process using `kill %2`.
  - If you have some stubborn GUI that won't die, use `xkill`.

Job Control (ctnd.)

- **To see all running processes, use the `ps` command.**
  - `ps` will list all process running in this terminal
  - `ps -u username` will list all the processes associated with a specific user
  - `ps -A` will show all running processes
- **For a graphical representation, use the `top` or `htop` commands**





- Sort jobs in `top` by memory consumption by pressing `shift-M`.
- Kill a job from `top` by pressing `k` and then entering the `PID`.

And some cool commands to finish

- **Sort words in a file with the `sort` command:**

```
$ sort names.txt
```

```
$ sort -r names.txt
```

- **Get rid of duplicates with the `uniq` command:**

```
$ uniq names.txt
```

```
$ sort names.txt | uniq
```

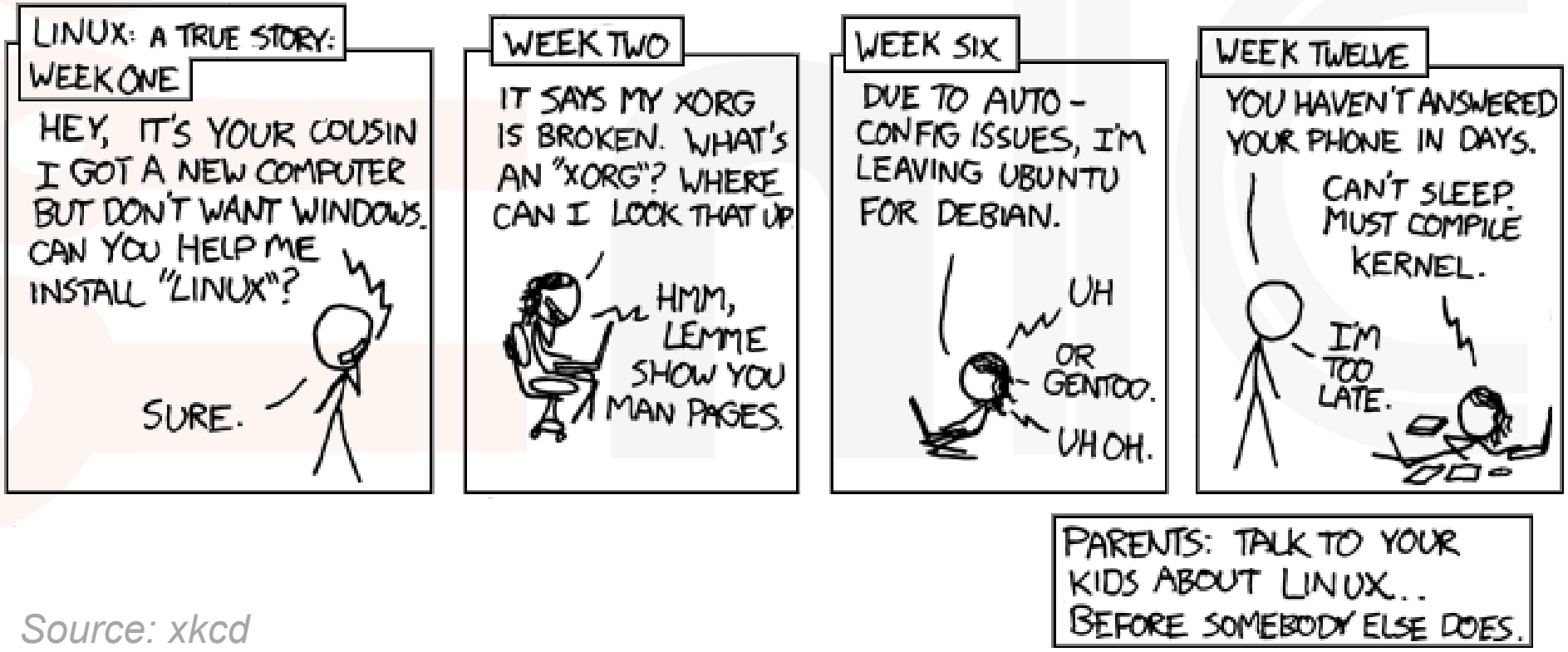
- **Count the number of words in a file:**

```
$ wc names.txt
```

```
$ sort names.txt | uniq | wc
```

The End

- **But really, it's just the beginning…**



Source: xkcd