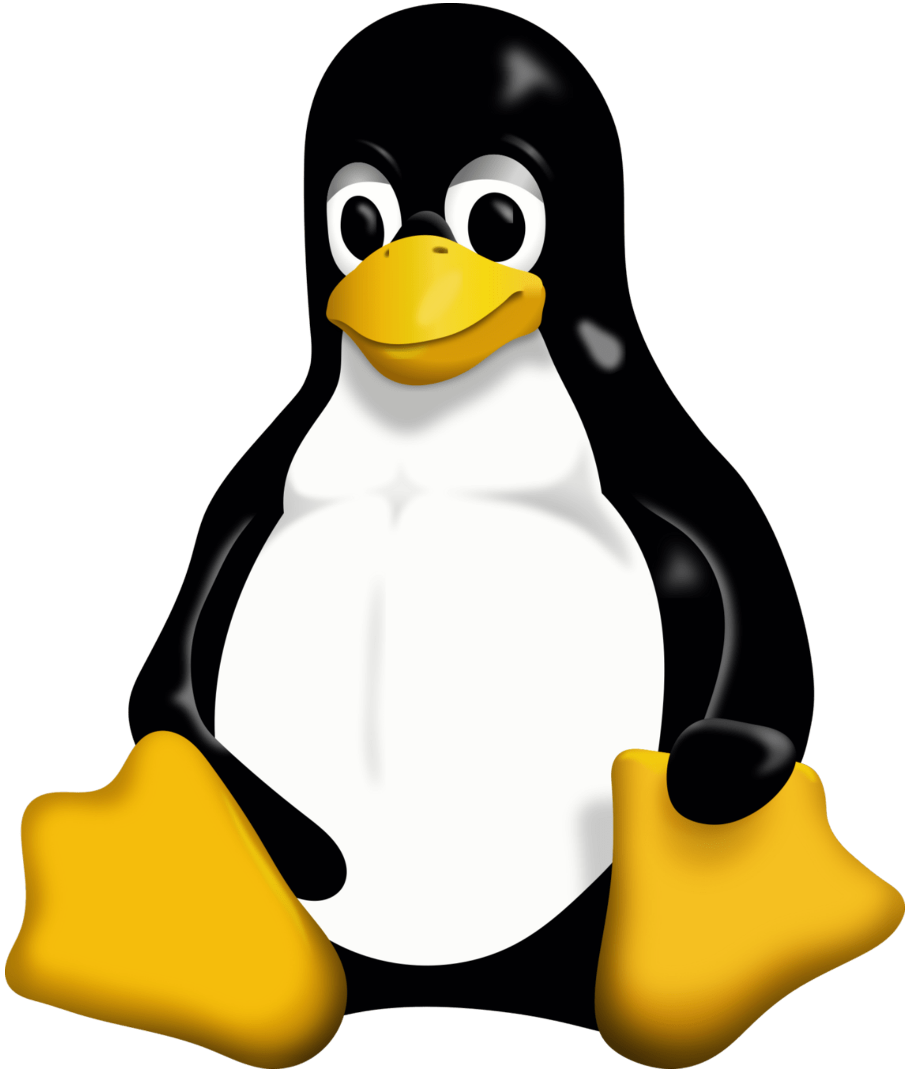
Linux Basics Course

May 16, 2024



WAGENINGEN
UNIVERSITY & RESEARCH

Instructors

- Atam Panday
- Dapeng Sun
- Jeremie Vanderplas
- Leonardo Honfi Camilo
- Nick Brummans

After this course, you should

- Have a basic understanding of the Linux operating system
- Be able to execute commands in bash
- Be able to write, change permissions and execute scripts
- Have the required knowledge to attend the container and HPC Basic courses

Ice Breaker

1. Go to **wooclap.com**
2. Enter code **WURLINUX**

Agenda

09:10 - Introduction to Linux

09:15 - Connecting to the HPC

09:30 - Bash Shell

09:45 - Navigating Files and Directories

10:15 - Break

10:30 - Working With Files and Directories

11:00 - Pipes, Filters and Redirects

11:15 - Loops

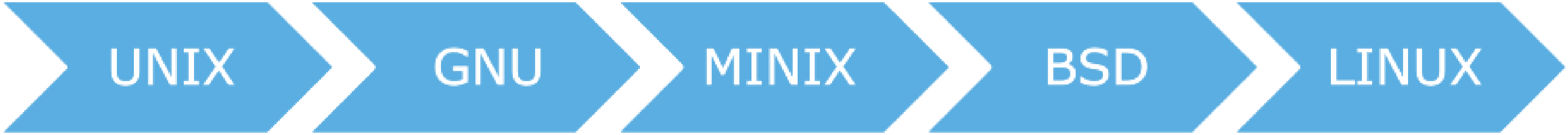
11:30 - Shell Scripts

12:30 - End

WAGENINGEN
UNIVERSITY & RESEARCH

Introduction to Linux

History



What is Linux?

Kernel

Manages hardware resources and provides essential services

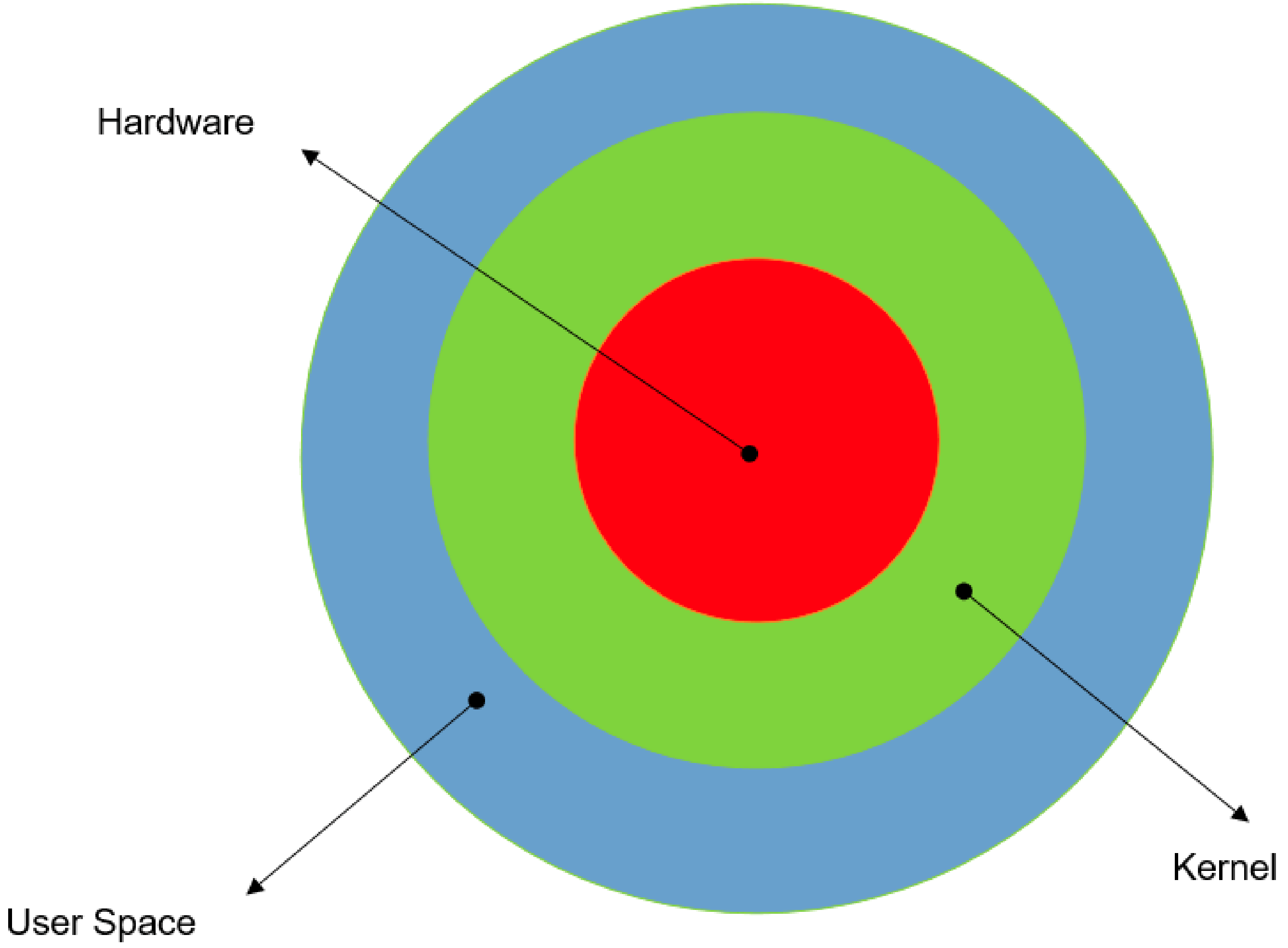
Operating System

A Linux **distribution** bundles the Linux kernel, system utilities, libraries, and often a package manager, to form an operating system

Linux is Everywhere

- Backbone of the internet
- Android phones
- IoT devices
- Cars
- Supercomputers
- Mars
- …

Linux Systems



Connecting to the WUR HPC

Guiding users on connecting to the HPC from Linux, Mac, and Windows.

Before You Connect

- The connection to the HPC is enabled by the Secure Shell (SSH) protocol
- On Linux and macOS, SSH is either packages or preinstalled.
- On Windows, we recommend the use of **MobaXTerm**
- If not, then you can use
  - Putty
  - Windows Subsystem for Linux (WSL)
  - PowerShell

WARNING: If you mistype the correct password 3 times, you account will be locked.

Connecting to the Anunna HPC

Open a terminal and run the command:

```
$ ssh username@login.anunna.wur.nl
```

WARNING: no characters are displayed when typing your WUR password!!!

MobaXTerm: go to Session => SSH and under remote host type fill in

```
login.anunna.wur.nl
```

The Bash Shell

Compilers vs Interpreters

Compilers:

Converts entire programs into executable machine code before execution.

- Faster execution speed due to pre-compiled code. Better optimization for performance.
- Higher complexity. Potentially, Harder to debug.

e.g. C/C++, Fortran, Java

Interpreters

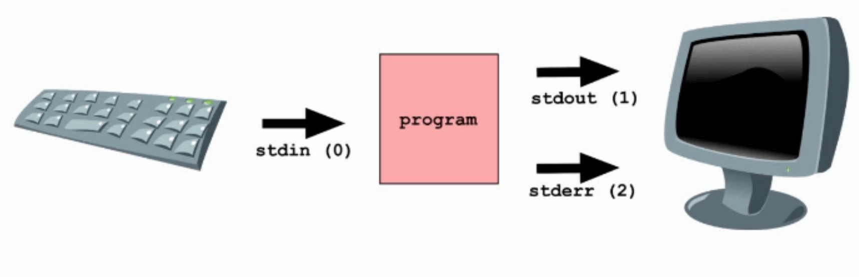
Translates high-level code into an intermediate form and executes it line-by-line.

- Good at Error Handling. Easier to debug, as it stops at the first error encountered.
- Slower execution speed.

e.g. Python, R, **bash**, zsh

Shells

- Basically, interactive interpreters, it has its own language syntax
- Runs in the user-space, on top of the kernel
- Accessible via "terminals" or terminal emulators.
- There are many shells out there.
- Comprised of three fields
    - Standard Input - What you type
    - Standard Output - What is print on screen in case of successful
    - Standard Error - What is print on the screen in case of failure



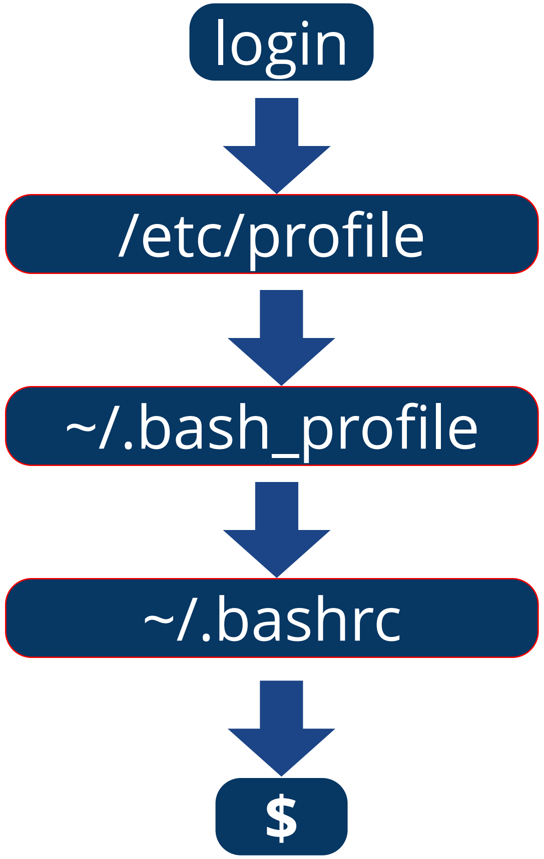
Bash shell - $

Bourne Again Shell

- Command-line interface that allows users to interact with the operating system by typing commands to perform operations and manage files and programs.
- Most popular, though there are many alternatives
- Interpreter located at **/bin/bash**
- It has its own language syntax
- Commands usually follow the format:

```
user001@login2:~$ <application> --<flag>/-<flag> <argument>
```

Shell Start-up



login

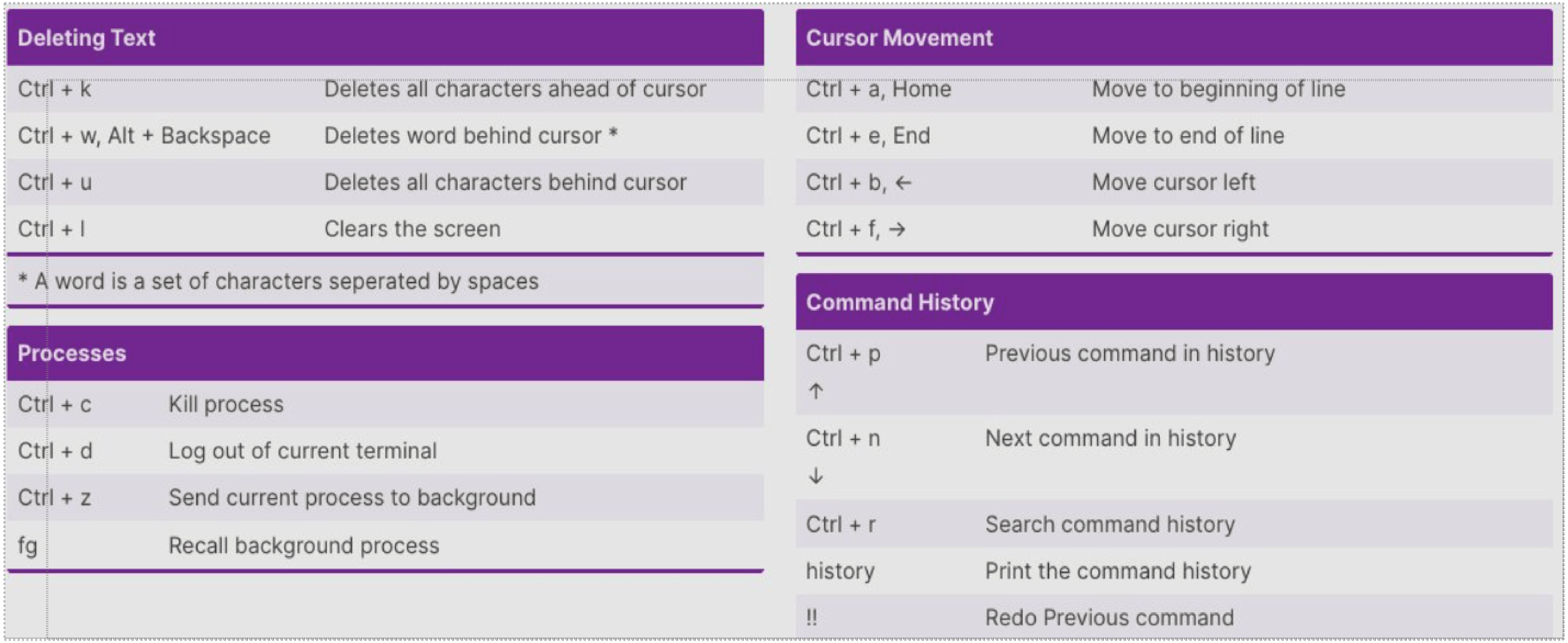
/etc/profile

~/.bash_profile

~/.bashrc

$

Keyboard Shortcuts



Deleting Text

Ctrl + k	Deletes all characters ahead of cursor
Ctrl + w, Alt + Backspace	Deletes word behind cursor *
Ctrl + u	Deletes all characters behind cursor
Ctrl + l	Clears the screen

* A word is a set of characters seperated by spaces

Processes

Ctrl + c	Kill process
Ctrl + d	Log out of current terminal
Ctrl + z	Send current process to background
fg	Recall background process

Cursor Movement

Ctrl + a, Home	Move to beginning of line
Ctrl + e, End	Move to end of line
Ctrl + b, ←	Move cursor left
Ctrl + f, →	Move cursor right

Command History

Ctrl + p ↑	Previous command in history
Ctrl + n ↓	Next command in history
Ctrl + r	Search command history
history	Print the command history
!!	Redo Previous command

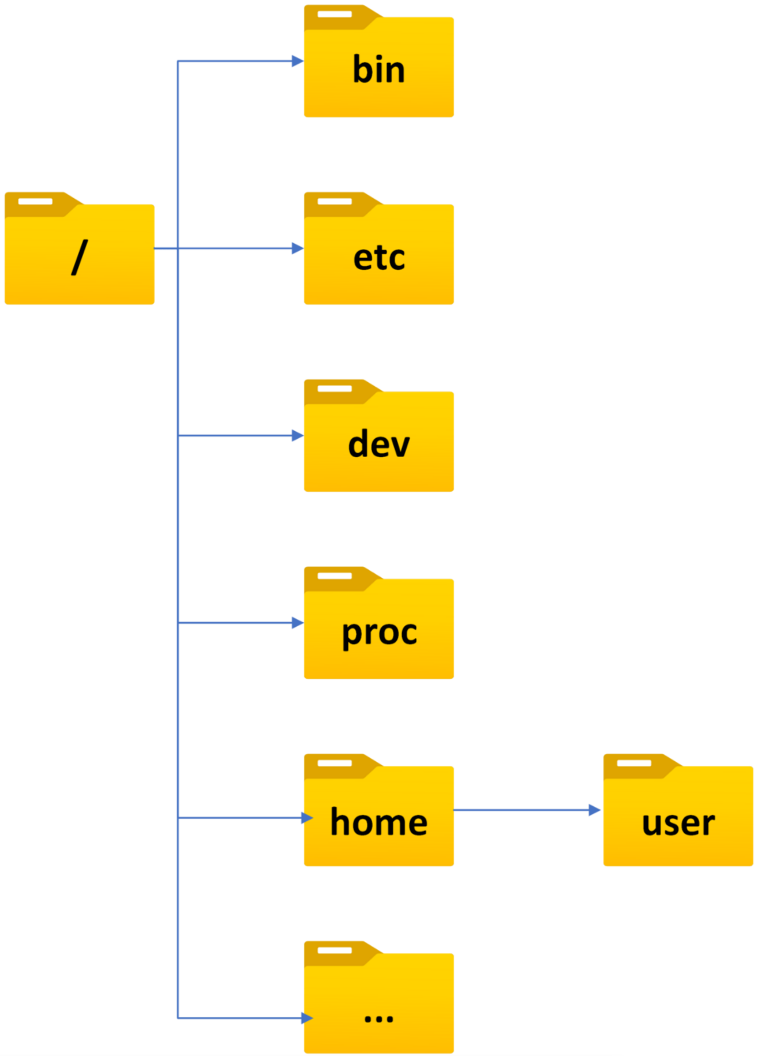
cheatography.com

Navigating Files and Directories

Typical Folder Structure

- Files are ordered in tree structure
- The root directory (**/**) is at the top of the tree
- **/bin**, **/sbin** and **/usr/bin** contain executable applications
- **/etc** contains the configuration files of the system
- **/dev** contains the files corresponding to devices
- **/proc** contains files corresponding to CPUs and GPUs
- **/tmp** contains temporary files
- **/opt** is a directory used to install optional software
- **/home** contains the folders corresponding to every user
- At the Anunna HPC the user's folders are a bit different, type:

```
$ pwd
```



```
/home/WUR/user001
```

cd - change directory

Template:

```
$ cd <directoryPath>
```

Go to home directory:

```
$ cd ~
```

Go to previous directory:

```
$ cd -
```

Go one directory up:

```
$ cd ..
```

Go two directories up:

```
$ cd ../..
```

relative path:

```
$ cd ./apps
```

full path:

```
$ cd /home/WUR/user001/apps
```

echo - Display Text

Template:

```
$ echo -<flags> <string>
```

Display contents of $PATH:

```
$ echo $PATH
```

Display string with escape characters:

```
$ echo -e "\nThis was a triumph!\n"
```

ls - List

Template:

```
$ ls -<flags> <fileOrDirectory>
```

list all files at the home directory:

```
$ ls -a ~
```

list all files in long format at the current directory, organizing with respect to time and present human readable file sizes

```
$ ls -alth .
```

Getting Help

Using the -h/--help flags

```
$ ls -h
```

```
$ ls --help
```

man - Manual Pages

Template:

```
$ man -<flags> <application>
```

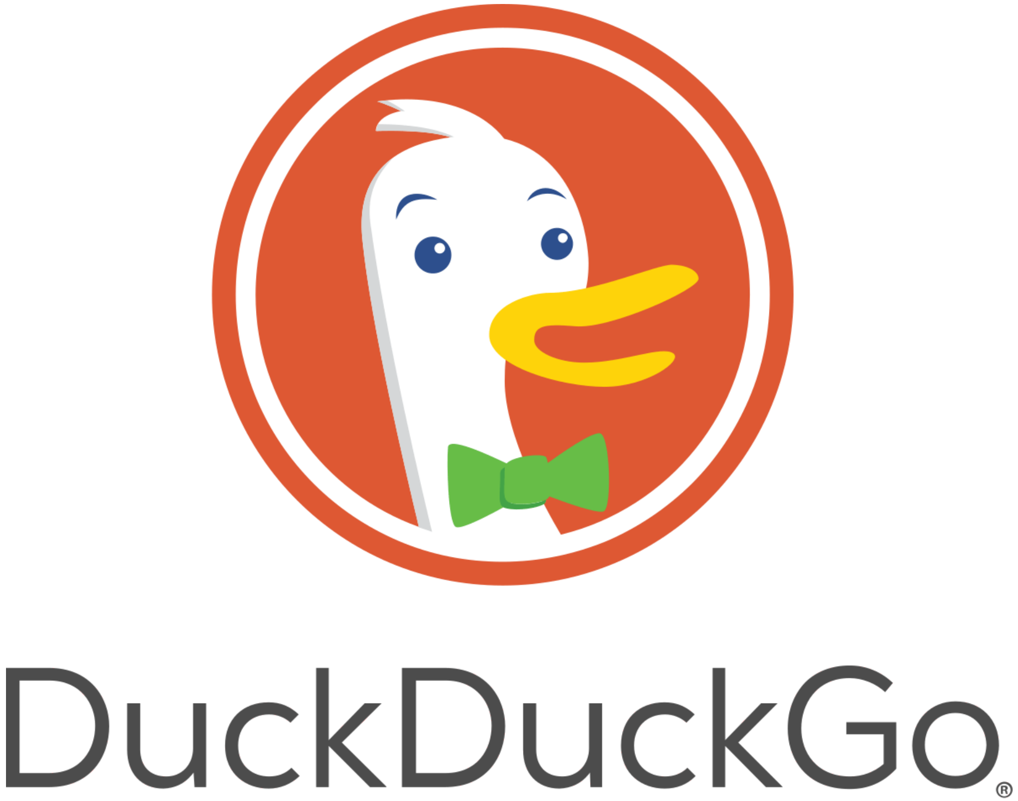
Manual pages of ls:

```
$ man ls
```

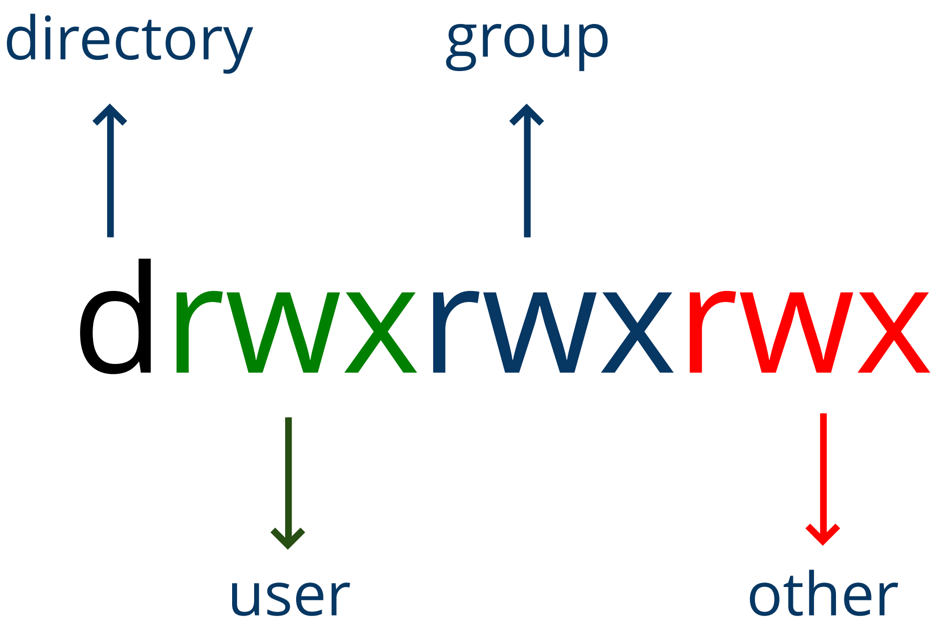
shortcuts

```
navigation: arrow keys, page down, page up
page down: space bar
search: / (n: previous, N: next)
quit: q
```

Alternatively



Permissions



directory

group

drwxrwxrwx

user

other

WAGENINGEN
UNIVERSITY & RESEARCH

Permission Octals

r : 2^2 w : 2^1 x : 2^0

rwx : $1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 4 + 2 + 1 = 7$

r-x : $1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 0 + 1 = 5$

r-- : $1 * 2^2 + 0 * 2^1 + 0 * 2^0 = 4 + 0 + 0 = 4$

Putting It All Together

rwxr-xr-x = 755

Exercise: Permission Octals - (5 min)

rw-r-xr-x

rw-r-----

rw-rw-rw-

rwxrwxrwx

Exercise: Permission Octals - solution

rw-r-xr-x = 655

rw-r----- = 640

rw-rw-rw- = 666

rwxrwxrwx = 777

Exercise - 10 min

Using the commands we just introduced, **ls** and **cd**, find the course files **shell-lesson-data.zip** at:

```
$ /lustre/shared/hpcCourses
```

- How big is the file?
- When was the file last modified
- What user owns the file
- What group owns the file
- What is the permissions octal of the file?

chmod - Changing Permissions

Template:

```
$ chmod  octal <file/folder> <optionFlags>
```

Changing permissions of a file

```
$ chmod 775 myfile
```

Changing permissions of a directory and all enclosed files and subdirectories

```
$ chmod 775 myDirectory -R
```

Making a file executable

```
$ chmod +x myfile
```

chown - Changing Ownership

Template:

```
$ chown  user:group <file/folder> <optionFlags>
```

Giving user001 ownership of myFile

```
$ chown  user001: myFile
```

Giving mygroup ownership of myDirectory without changing the user ownership

```
$ chown :mygroup myDirectory -R
```

Assigning onwership all the files and folders inside myDirectory to user001

```
$ chown user001:user001 myDirectory -R
```

Working With Files and Directories

mkdir - make directory

Template:

```
$ mkdir -<flags> <directoryPath>
```

Create folder at home directory:

```
$ mkdir ~/newFolder
```

Create folder with parents:

```
$ mkdir -p ./first/second/newFolder
```

cp - Copy

Template:

```
$ cp -<flags> <source> <target>
```

copying files (interactive):"

```
$ cp -i file01 file02
```

copying directories (interactive):

```
$ cp -ri directory01 directory02
```

mv - Move

Template:

```
$ mv -<flags> <source> <target>
```

Rename files (interactive)

```
$ mv -i ./file01 ./file02
```

Rename directory (interactive):

```
$ mv -i directory01 directory02
```

rm - Remove

Template:

```
$ rm -<flags> <fileOrDirectory>
```

Remove multiple files (interactive):

```
$ rm -i ./file01 ./file02 ./archive/file03.txt
```

Remove multiple directories (interactive):

```
$ rm -ri directory01 directory02
```

WAGENINGEN UNIVERSITY & RESEARCH

Exercise - Copy Data File - 5 min

location:

```
/lustre/shared/hpcCourses/shell-lesson-data.zip
```

At your home directory unzip the file with thecommand:

```
$ unzip ~/shell-lesson-data.zip
```

Use the tree command to explore the contents up to 2 levels

```
$ tree -L 2 ~/shell-lesson-data
```

Globbing - Wildcards

Wildcard characters used to match one of more filename characters

***** - Wildcard to one of more characters

```
ls *.txt
```

? - wilcard for a single character at a specific position

```
$ rm -i file0?.txt
```

? can be used as many times as necessary

```
$ rm -i file???.txt
```

Globbing - Ranges

Ranges are used usually for specific alphanumeric character

List all files containing a single digit

```
ls *[0-9]*
```

- [a-z] = all lowercase characters of the alphabet
- [A-Z] = all uppercase characters of the alphabet
- [a-zA-Z] = all characters of the alphabet, irrespective of their case
- [j-p] = lowercase characters j, k, l, m, n, o or p
- [a-z3-6] = lowercase characters or the numbers 3, 4, 5 or 6

List Ranges

Used to create a continuous alphanumeric sequence. Must be inside braces

Create empty files from 0 to 4

```
touch file{0..4}.txt
```

Some padding can be added

```
touch file{01..06}
```

Alphabet characters also can be included

```
echo folder_{a..e}
```

Exercise

- In your home directory create a folder labelled **temp**
- Inside **~/temp,** combine multiple ranges to create empty files (feel free to experiment)

```
touch ~/temp/file{0..3}{0..3}{a..e}.txt
```

- Use wildcards or/and globing ranges to clear the contents of the **temp** folder

- Use wildcards to find hidden files in your home directory. (**Hint**: hidden files always begin with a period ".")

Exercise - Solution

```
$ mkdir ~/temp
$ touch ~/temp/files{0..3}{0..3}{a..e}.txt
$ ls ~/temp
$ rm ~/files???.txt

$ ls ~/.*
```

Finding

Used to search for files and directories within a directory hierarchy based on various criteria.

Template:

```
$ find  <directoryPath> <optionFlags> <expression>
```

Common Options

- **-name** : Search by file or directory name
- **-type** : Filter by type (e.g., f for regular file, d for directory)
- **-exec** : Execute a command on the found items

Find Example

Find minotaur.dat at the data folder

```
$ find ~/shell-lesson-data -type f -iname minotaur*

./exercise-data/creatures/minotaur.dat
```

Exercise - unicorn - 1/2

- find a file called unicorn.dat in your home directory
- In your home (~) directory, create a directory called research, which contains another directory called unicorn
  - **Bonus:** Did you do this with one or multiple commands, how can this be done with a single command
- Change into the newly created unicorn directory

```
$ cd ~/research/unicorn
```

Exercise - unicorn - 2/2

- Create a copy of the unicorn.dat file, into your current working directory, using a relative path notation
- Rename the newly created copy of the file in the current path to unicorn-data.txt
- Using **ls** to look at the details of the file, has anything changed from the original besides the name
- If something did change, what step would have caused this and how could this have been prevented

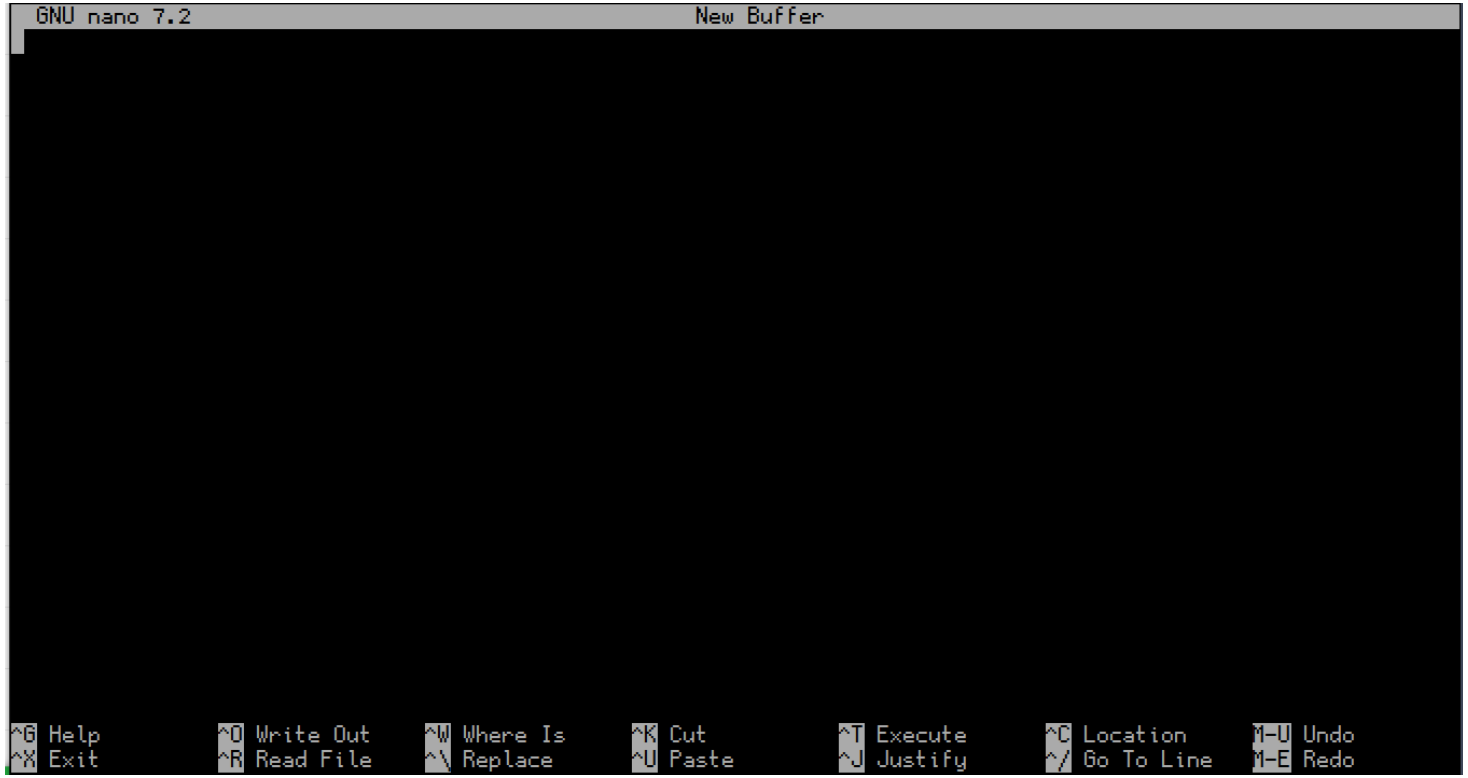
Text Editors

- Nano (recommended)
- Vim
- emacs
- VScode/pyCharm with Remote SSH Extension

WARNING: Only use VScode/pyCharm for editing files.

Do not use them to run scripts/jobs

GNU Nano



```
  GNU nano 7.2                               New Buffer




^G Help          ^O Write Out     ^W Where Is      ^K Cut           ^T Execute       ^C Location      M-U Undo
^X Exit          ^R Read File     ^\ Replace       ^U Paste         ^J Justify       ^/ Go To Line    M-E Redo
```

Cheatsheet:

- “Home” goes to front of line
- “End” goes to end of line
- Drag mouse over text
- Right click to copy
- Right click to paste
- “Ctrl + o” Write file
- “Ctrl + x” Exit nano

Tip:

^X means pressing the Control and X key

M-X means Meta/Windows key and X

Exercise - Editing with Nano - 1/2

- Open unicorn-data.txt in the nano text editor

```
$ nano ~/research/unicorn-data.txt
```

- With unicorn-data.txt open in nano do the following:
- On the first line change "COMMON NAME:" from unicorn to unicorn-data
- On the third line change the "Updated:" date to today
- Select the top three lines (**M-A** Mark Text) and press <**TAB**> to indent
- Now comment the three selected lines out
  - Hint: Search the help menu for the shortcut for "*Comment/uncomment the current line (or marked lines)*"

WAGENINGEN
UNIVERSITY & RESEARCH

Exercise - Editing with Nano - 2/2

- Find the first line in the file containing: **TACCGGACAA**
- Select the line and copy the text
- Search for more lines containing the same information
  - How many are there in total
  - What happens when you reach the end of the file

Cheatsheet:

^W – Where Is,

M-A – Mark Text,

M-6 – Copy Text,

^U – Paste Text,

M-Q – Previous,

M-W - Next

grep

Used to search for text patterns within files or command output.

Template:

```
grep <optionFlags> <pattern> <file>
```

Common Options:

- -i : Perform a case-insensitive search
- -v: Except.
- -r or -R : Recursively search directories
- -A, -B or -C : Display lines after, before, or around the matching lines

WAGENINGEN
UNIVERSITY & RESEARCH

wc - Word Count

Used to search for text patterns within files or command output.

Template:

```
wc <optionFlags>  <file>
```

Common Options:

- **-l, --lines** : Counts number of lines
- **-w, --words** : Counts number of words
- **-m, --chars** : Counts number of characters
- **-c, --bytes** : Counts number of bytes

Checking

We can use grep and wc to check the answer of the previous exercise

```
$ grep TACCGGACAA ~/research/unicorn/unicorn-data.txt | wc -l
```

Pipes, Filters and Redirects

Redirects >

- Given by the " **>** " operator
- Used to "redirect" the standard output to a file
- If a file does not exist, it will create it
- If a file already exists, it will overwrite it.
- To avoid overwrite, use " **>>** " to append

Pipe

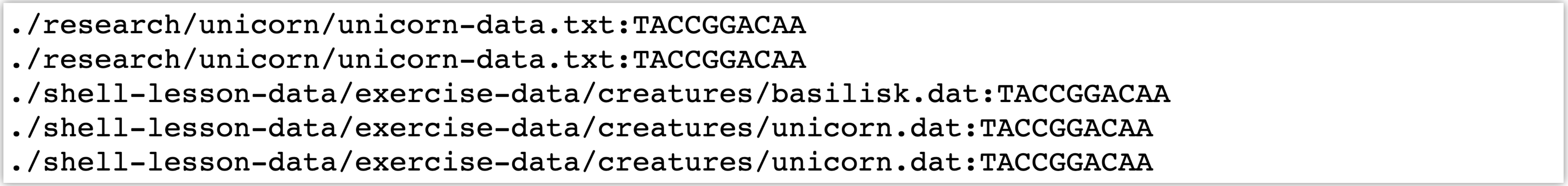
- Given by the "|" operator
- "Pipes" the output of one command into the input of another command
- Can be used multiple times to create complex pipelines

Useful tools

- **cat** - concatenates several files into a single output
- **wc** - counts lines, words, characters or bytes
- **head** - Displays the first N lines (default: 10)
- **tail** - Displays the last N lines (default: 10)
- **tr** - "translates"/replaces patterns
- **cut** - cuts strings wrt delimiters
- **uniq** - reports or omits repeated lines
- **sort** - sorts the content of a file

Exercise - pipes and filters - 1/2

- Using the grep command find all files in your home (~) directory containing the text TACCGGACAA ignoring case
- The result should something like the below
- Run the command again, this time redirect the output (using >) to a file in your home directory called redirected.txt
- Using the cat command, output the content of ~/redirected.txt to your screen



```
./research/unicorn/unicorn-data.txt:TACCGGACAA
./research/unicorn/unicorn-data.txt:TACCGGACAA
./shell-lesson-data/exercise-data/creatures/basilisk.dat:TACCGGACAA
./shell-lesson-data/exercise-data/creatures/unicorn.dat:TACCGGACAA
./shell-lesson-data/exercise-data/creatures/unicorn.dat:TACCGGACAA
```

Exercise - pipes and filters - 2/2

- Pipe the output into another command that counts the number of occurrences and outputs a number
- Were you expecting this number after performing the initial grep command
- Run the command again, this time append(using >>) the output again to redirected.txt
- How can we count the actual number of files containing the text, ignoring multiples occurrences per file
- Run the command again, this time append (using >>) the output again to redirected.txt

Loops

Types

For

Iterate over a list of items

While

Iterate **while** a condition is true

Until

Iterate **until** a condition is met

For Loops

Template:

```
$ for user in john mary sarah
>do
>echo Hello, $user
>done
```

Inline:

```
$ for user in john mary sarah ;do echo Hello, $user;done
```

WAGENINGEN
UNIVERSITY & RESEARCH

while/Until Loops

```
$ while [condition]
>do
>echo "Still running"
>done
```

```
$ until [condition]
>do
>echo "Still running"
>done
```

If Statement

```
$ if [condition]; then
> <commands>
> fi
```

Exercise - Loops 1/2

- Change directory to ~/shell-lesson-data/exercise-data/alkanes
- Have a look at the below BASH one liner loop command and try and reason what is will do

```
$ for file in *; do if [ -f "$file" ]; then echo -n "File found: $file, lines: ";
wc -l < "$file"; fi; done
```

- Now actually run the command and see if you were correct
- Do you think this loop is easy to read?

Exercise - Loops 2/2

- The same command can also be written across multiple lines, which would look like this

```
 $ for file in /path/to/directory/*; do \
    if [ -f "$file" ]; then \
        echo -n "File found: $file, Lines: "; \
        wc -l < "$file"; \
    fi; \
done
```

- The \ at the end of the line indicates that the command continues on the next line
- This makes it more readable for most people, but hard to edit on the command line
- Creating a script solves this issues and we'll discuss these next

Shell Scripts

Scripts

```
#!/bin/bash

echo -e "\nHello, $USER\n"
```

- Collection of commands
- Executed in sequence (top to bottom)
- First line of the script defines interpreter (#!)
- Must be executable (permissions)

Alternative Interpretors

For python scripts:

```
#!/bin/env python
```

For R scripts:

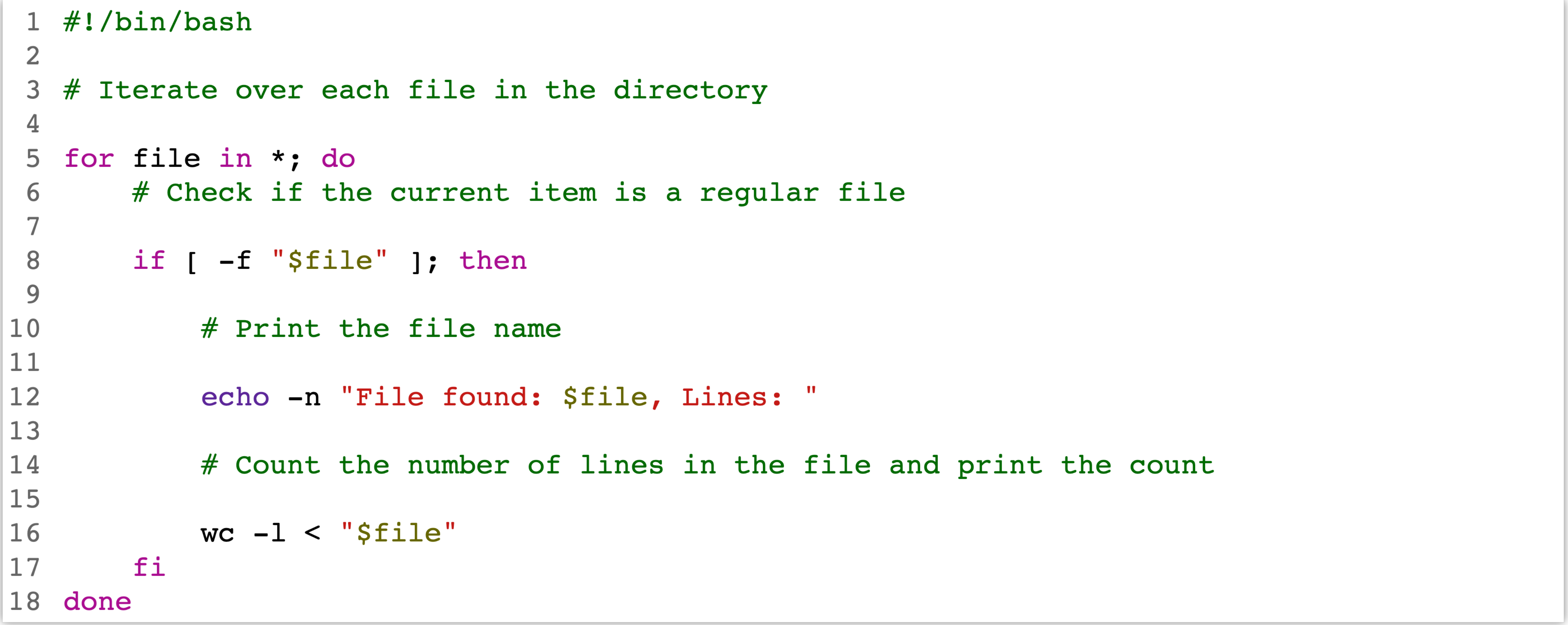
```
#!/bin/env Rscript
```

Making Scripts Executable

```
user001@login2:~$ ls -l hello.sh

-rw-r--r-- 1 user001 domain users 0 Apr 16 06:52 hello.sh

user001@login2:~$ chmod +x hello.sh

user001@login2:~$ ls -l hello.sh

-rwxr-xr-x 1 user001 domain users 0 Apr 16 06:52 hello.sh
```

Exercise - Writing Scripts - 1/3

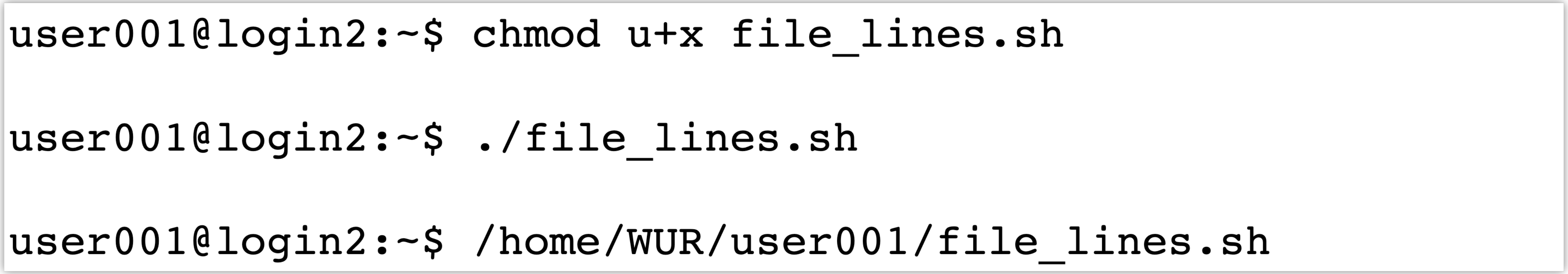
- Let's take the BASH one liner we used as a loop and create a script called file_lines.sh



```bash
#!/bin/bash

# Iterate over each file in the directory

for file in *; do
    # Check if the current item is a regular file

    if [ -f "$file" ]; then

        # Print the file name

        echo -n "File found: $file, Lines: "

        # Count the number of lines in the file and print the count

        wc -l < "$file"
    fi
done
```

Exercise - Writing Scripts - 2/3

- Make the script executable and run it



```
user001@login2:~$ chmod u+x file_lines.sh

user001@login2:~$ ./file_lines.sh

user001@login2:~$ /home/WUR/user001/file_lines.sh
```

- What is the difference between the last two commands above?

Exercise - Writing Scripts - 3/3

- Create a directory in your home called **apps**
- Move files_lines.sh into the **apps** directory
- Does it work this time? why?

```
$ mkdir ~/apps

$ mv ~/file_lines.sh apps/

$ file_lines.sh
```

Environment variables

- Bash environment variables are key-value pairs stored within the Bash shell that influence the behaviour of software on the system.
- Environment variables provide a way to customize the system's behaviour, specify default settings for applications, and simplify interactions between different components of the system.
- They can be used to configure shell settings, store data like paths to executables or directories, and control the operation of scripts and applications.

Env and Notable Variables

Display the variables in your session:

```
$ env
```

Notable:

- **HOME** - stores the location of your home directory
- **PATH** - stores locations of your executable files (separated by :)
- **LD_LIBRARY_PATH** - stores locations of libraries
- **MODULEPATH** - Stores the location of the system modules

Note: Environment variables are presented in higher case.

Creating environment variables

You can create your own variables

```
myVariable="Hello"

export myOtherVariable="Hello"
```

Optional - Start scripting - 1/2

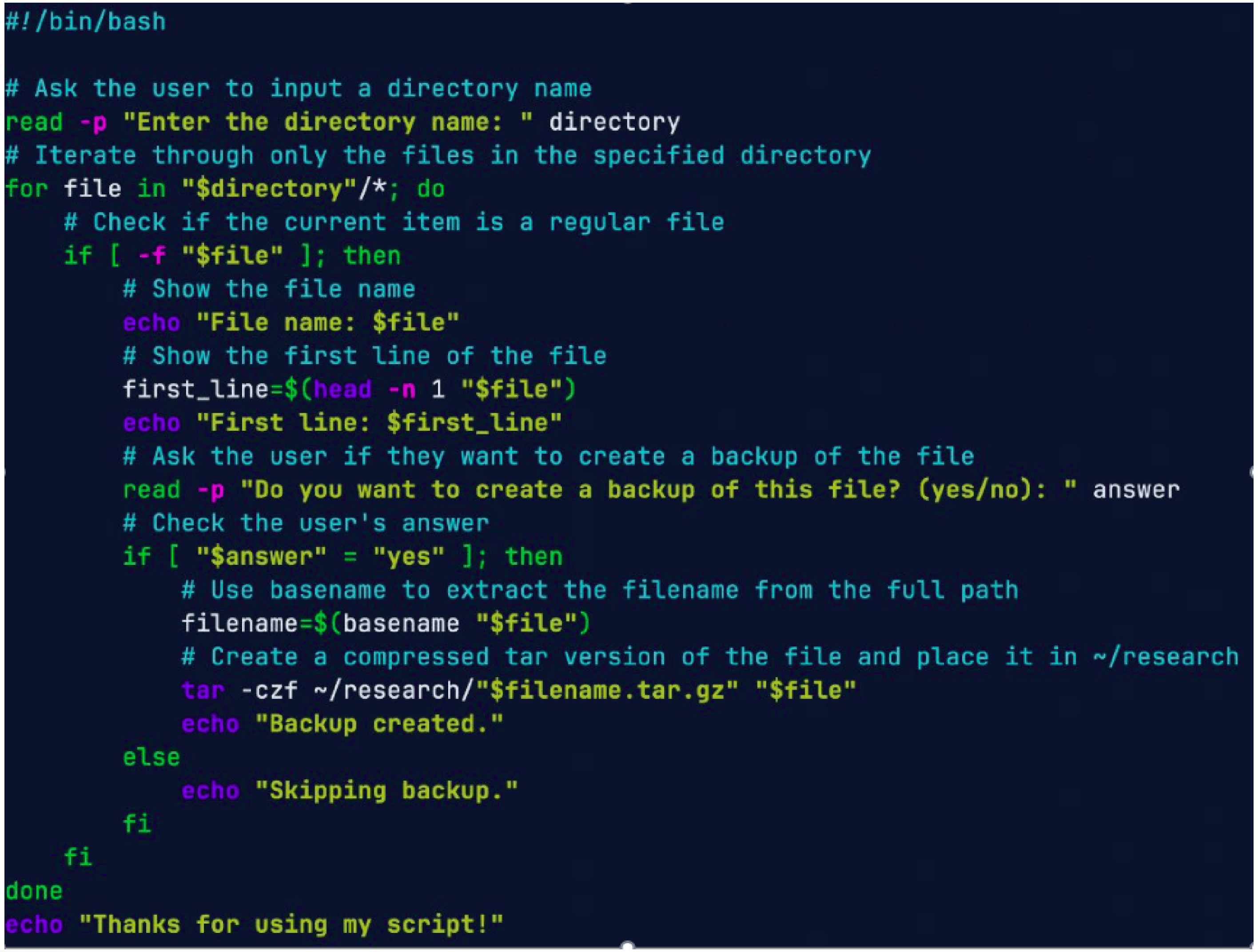
- Create a script that does the following
- Ask the user to input a directory name
- The script should then iterate through only the files in the specified directory
- Show the files name
- Show the first line of the file
- Ask the user if they want to create a backup of the file
- If you answer yes, the script should create a compressed tar version of it and place it in ~/research
- If the answer is no, continue with the next file
- When all files have been evaluated, the script ends with the message "Thanks for using my script!"
- Provide inline comments explaining the purpose of each command or section of the script
- Test the script on different directories

Optional - Start scripting - 2/2

Hints

- Use nested if-then statements
- read -p "Enter some information: " variable   # Ask the user for input and store in variable
- print $variable   # Shows the content of the variable
- filename=$(basename "$file")   # Strip the filename from a full path, store it in variable filename
- **Bonus:** Check if the entered directory actually exists, before proceeding otherwise exit

Optional – Possible Solution



```bash
#!/bin/bash

# Ask the user to input a directory name
read -p "Enter the directory name: " directory
# Iterate through only the files in the specified directory
for file in "$directory"/*; do
    # Check if the current item is a regular file
    if [ -f "$file" ]; then
        # Show the file name
        echo "File name: $file"
        # Show the first line of the file
        first_line=$(head -n 1 "$file")
        echo "First line: $first_line"
        # Ask the user if they want to create a backup of the file
        read -p "Do you want to create a backup of this file? (yes/no): " answer
        # Check the user's answer
        if [ "$answer" = "yes" ]; then
            # Use basename to extract the filename from the full path
            filename=$(basename "$file")
            # Create a compressed tar version of the file and place it in ~/research
            tar -czf ~/research/"$filename.tar.gz" "$file"
            echo "Backup created."
        else
            echo "Skipping backup."
        fi
    fi
done
echo "Thanks for using my script!"
```

Closing Remarks

Links For Self Study

Linux Journey

Software Carpentry

So long, and thanks for all the fish!

WAGENINGEN
UNIVERSITY & RESEARCH