Linux Basics Course

17 February 2026

WAGENINGEN
UNIVERSITY & RESEARCH

100years
1918 — 2018

Instructors

- Lukas Pellegrini
- Jeremie Vandenplas
- Leonardo Honfi Camilo

After this course, you should

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

Ice Breaker

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

WAGENINGEN
UNIVERSITY & RESEARCH

Agenda

- 00 Introduction to Linux
- 01 Connecting to the HPC
- 02 Bash Shell
- 03 Navigating Files and Directories
- 04 Break
- 05 Working with Files and Directories
- 06 Pipes, Filters and Redirects
- 07 Shell Scripts
- 08 Closing Remarks
- 09 Extra: Loops

Introduction to Linux

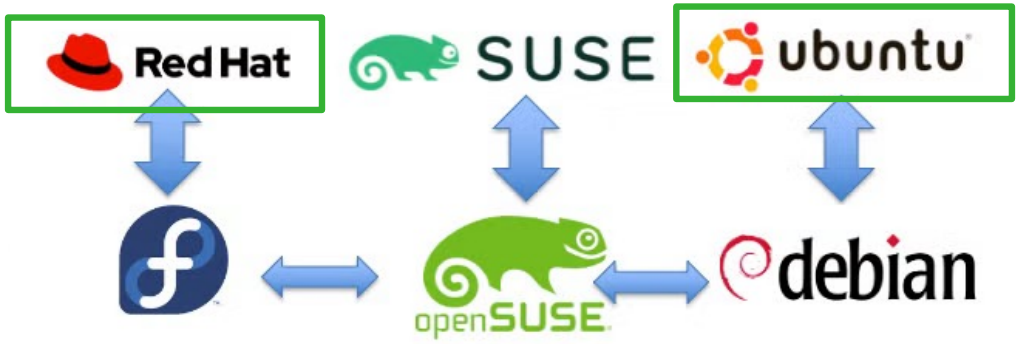
UNIX → GNU → MINIX → BSD → LINUX

Ken Thompson and Dennis Ritchie

Linus Torvalds

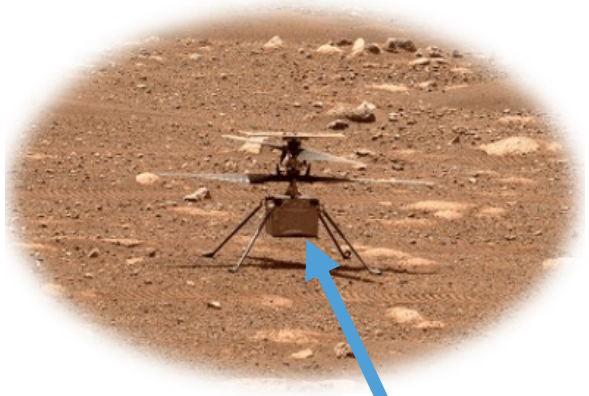
What is Linux?

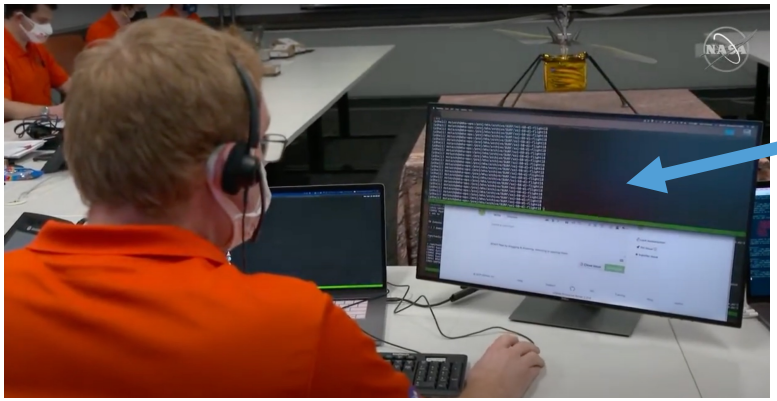
- Kernel
  - Manages hardware resources and provides essential services
- Operating System
  - A Linux distribution bundles the Linux kernel, system utilities, libraries, and often a package manager, to form an operating system



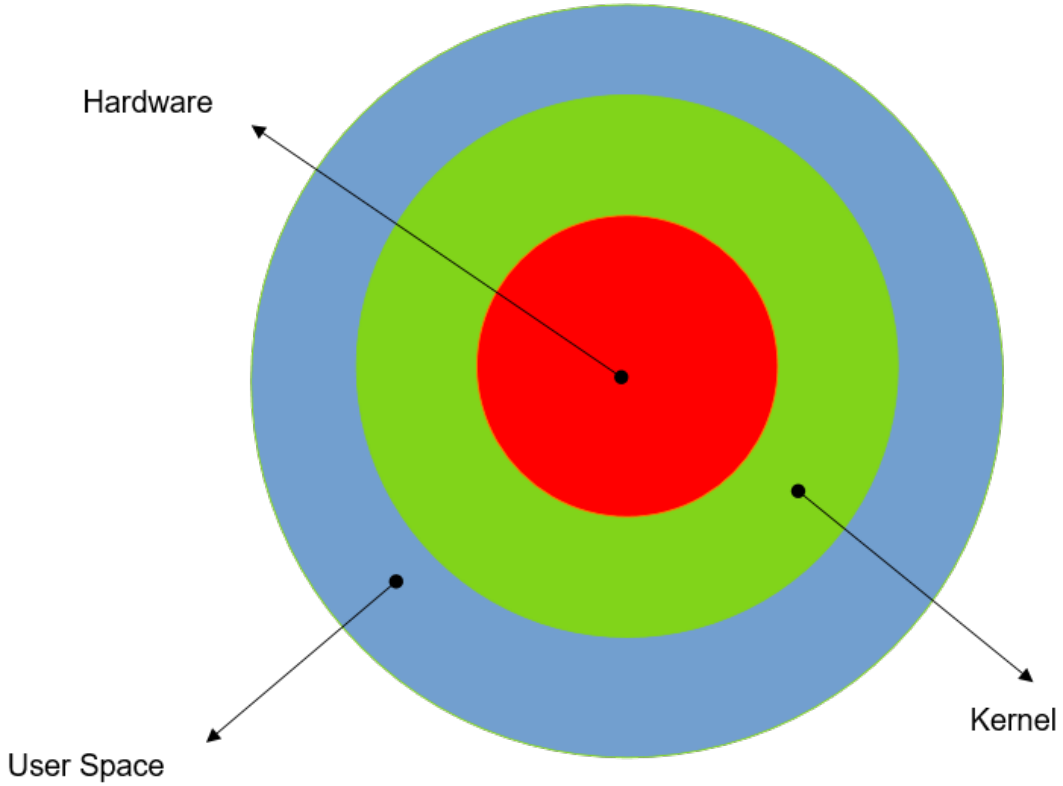
Linux is Everywhere

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





Linux Systems



Connecting to the WUR HPC (Anunna)

- Local connect over terminal
  - Windows Terminal
  - WSL
  - MobaXTerm
  - Mac Terminal
- Over the web browser
  - https://ood.anunna.wur.nl

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

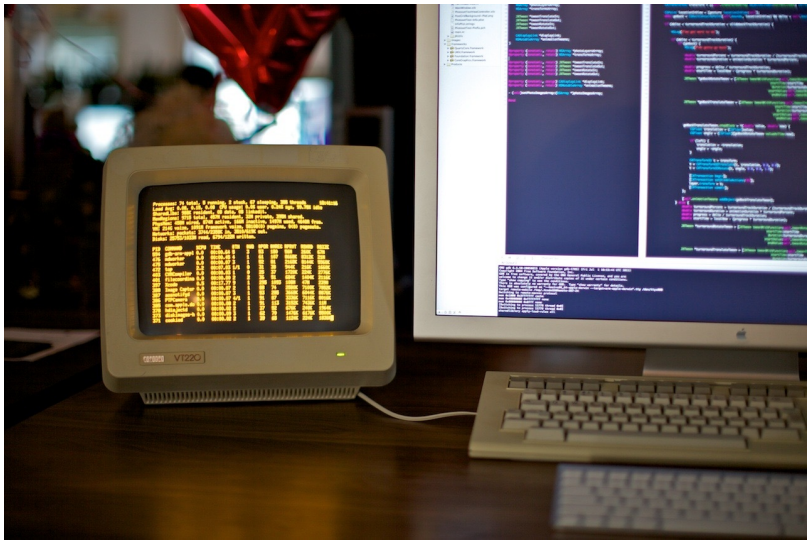
Connecting to the Anunna HPC via SSH

- Open the terminal and run this command

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

WARNING: No characters are displayed when typing your WUR password

The Bash Shell



Compilers vs Interpreters

Compilers

Converts entire programs into executable machine code before execution.

- Faster execution speed due to precompiled code. Better optimization for performance
- Higher complexity. Potentially, harder to debug
- E.g. C/C++, Fortran, Java

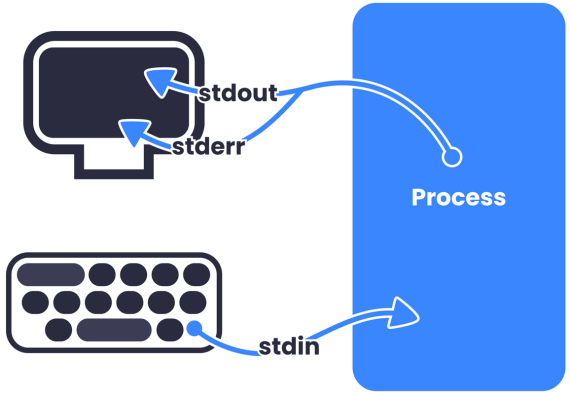
Interpreters

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

Shell

- 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: TEXT
- There are many shells out there
- Comprised of three fields
    - Standard Input – What you type
    - Standard Output– What is printed on screen in case of success
    - Standard Error – What is printed on the screen in case of failure



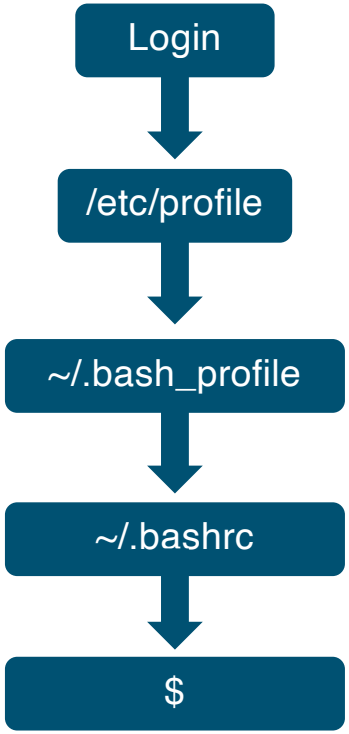
Bash Shell - $

Bourne **A**gain **S**hell

- 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:

```
user01@login201:~$ <app> --<flag>/-<f> <argument>
```

Shell Start-up



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

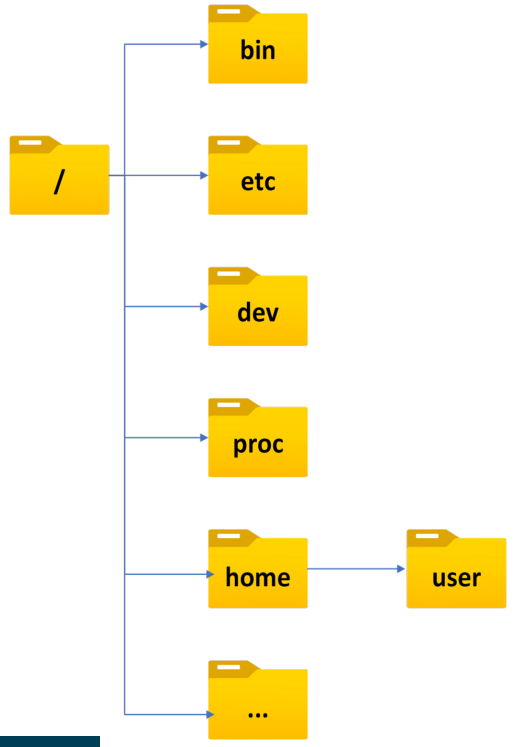
Navigating Files and Directories

Typical Folder Structure

- 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
- **/tmp** contains temporary files
- **/opt** is a directory used to install optional software
- **/home** ~ contains the folders corresponding to every user
- At Anunna:

```
$ 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

```
$ cd -h
```

```
$ cd --help
```

Man – Manual Pages

Template

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

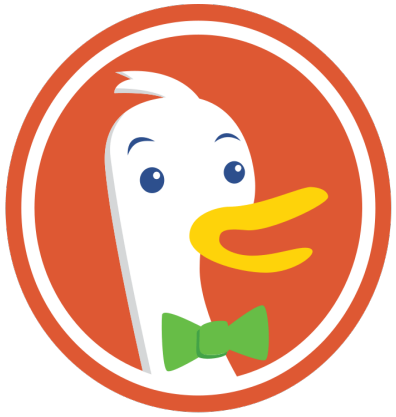
Manual pages of ls

```
$ man ls
```

shortcuts

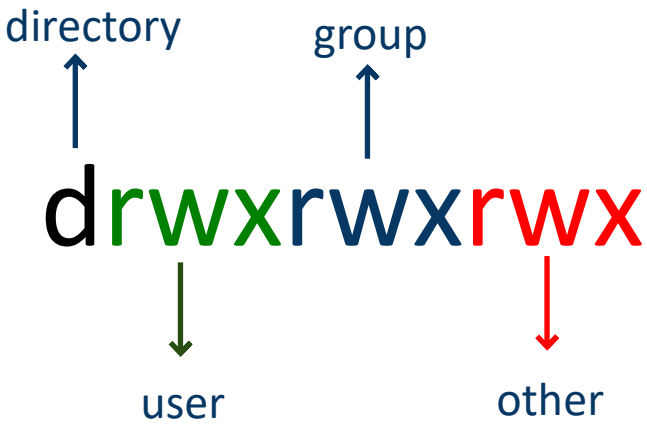
```
Navigation: arrow keys, page down, page up
Page down: space bar
Search: / (n: previous, N: next)
Quit: q
```

Alternatively



DuckDuckGo®

Permissions



Permission Octals

$$r : 2^2 \quad w : 2^1 \quad x : 2^0$$

$$\text{rwx} : 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 4 + 2 + 1 = 7$$

$$\text{r-x} : 1 * 2^2 + 0 * 2^1 + 1 * 2^0 = 4 + 0 + 1 = 5$$

$$\text{r--} : 1 * 2^2 + 0 * 2^1 + 0 * 2^0 = 4 + 0 + 0 = 4$$

Putting It All Together

rwxr-xr-x = 755

Exercise: Permission Octals – (5 min)

rwxr-xr-x =

rw-r----- =

rw-rw-rw- =

rwxrwxrwx =

Exercise: Permission Octals – solution

rwxr-xr-x = 655

rw-r----- = 640

rw-rw-rw- = 666

rwxrwxrwx = 777

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
```

Making a file executable

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

Exercise – 10min

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 Is the owner of the file?
- What group owns the file?
- What is the permissions octal of the file?

Let’s have a break

```
$  if [ \$FOCUS -lt 10 ]; \
   then echo 'Need a break!’; \
   break; \
   fi
```

BREAK SLIDE

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 directories (interactive)

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

rm - Remove

Template

```
$ rm –<flags> <fileOrDirectory>
```

Remove files (interactive)

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

Remove directories (interactive)

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

Globbing - Wildcards

Wildcard characters used to match one of more filename characters

* – Wildcard to one of more characters

```
$ ls *.txt
```

? – wildcard 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 **~** directory, create a folder labelled *temp*
- Inside ***~/temp***, combine multiple ranges to create empty fields (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 **~** 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 ~/.*
```

Exercise

- Copy the course files shell-lesson-data.zip in your **~** directory:

```
$ cp /lustre/shared/hpcCourses/shell-lesson-data.zip ~/
```

- Extract the archive:

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

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

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

Exercise – unicorn – 1/2

- Find a file called unicorn.dat in your **~** directory
- In your **~** directory, create a directory ***research***, which contains another directory ***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 file ***unicorn.dat*** 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?

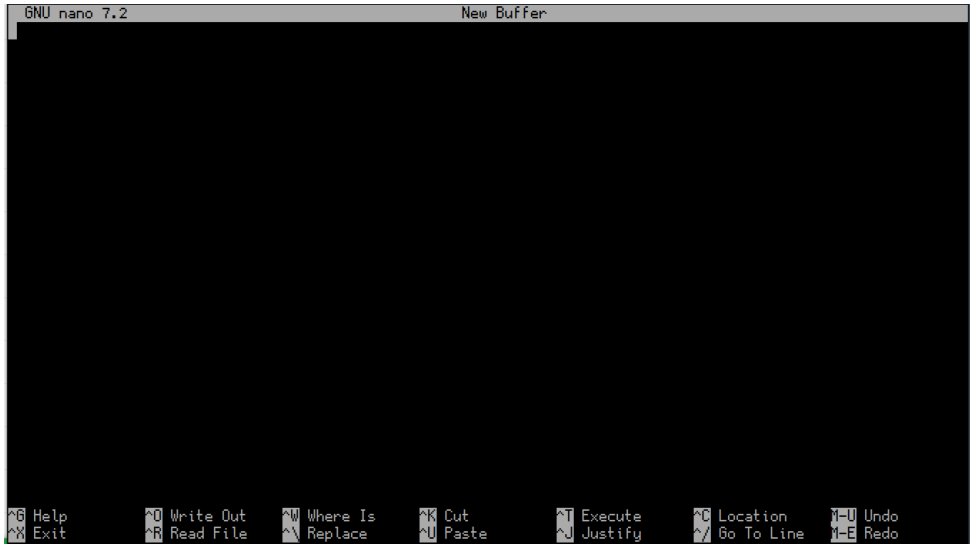
Text Editors

- Nano (recommended for beginners)
- Vim/neoVim
- 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



Tip:

^X means pressing the Control and X key

M-X means Meta (usually the Alt key) and X

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

WAGENINGEN UNIVERSITY & RESEARCH

100years 1918 — 2018

Exercise – Editing with Nano – 1/2

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

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

- 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
    - Note: some of the keybindings will not work in the web terminal
  - Comment the three selected lines out
    - Hint: Search the help menu for the shortcut for "Comment/uncomment the current line (or marked lines)"

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-insentive search
- -v: Except
- -r or -R: Recursively search directories
- -A, -B or –C: Display lines after, before, or around the matching lines

wc – Word Count

- Used to display the count of words/lines

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
```

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 **~** directory containing the text TACCGGACAA
- The result should be something like the below:

```
./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
```

- Run the command again, this time redirect the output (using >) to a file in your **~** directory called **redirected.txt**
- Using the cat command, output the content of **~/redirected.txt** to your screen

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 multiple occurrences per file?

Shell Scripts

Scripts

```
1 #!/bin/bash
2
3 echo -e "\nHello, $USER\n"
```

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

Alternative interpretors

For python scripts

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

For R scripts

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

Making Scripts Executable

```
user001@login201:~$ ls -l hello.sh

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

user001@login201:~$ chmod +x hello.sh

user001@login201:~$ 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 a loop and create a script called file_lines.sh

```
#!/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@login201:~$ chmod u+x file_lines.sh

user001@login201:~$ ./file_lines.sh

user001@login201:~$ /home/WUR/user001/file_lines.sh
```

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

Exerise – Writing Scripts – 3/3

- Create a directory in your **~** 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

- In Bash **environment vararibles** are key-value pairs stored within the Bash shell that influence the behaviour of software on the system.
- Env var provide a way to:
    - customize the systems's behaviour,
    - specify default settings for applications, and
    - simplify interactions between different components of the system.

Environment variables

- 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
```

Notables

- **HOME** – stores the location of your ~ 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 env vars

- You can create your own variables

```
myVariable="Hello"
```

- The export command makes the variable available in the entire bash session

```
export myVariable
export myOtherVariable="Hello"
```

Closing Remarks

- Feedback form: https://forms.office.com/e/CbPxn4PGYi

Links For Self Study

Linux Journey

Software Carpentry

So long, and thanks for your attention

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;do
```

While/Unit 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 it 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 issue and we'll disucss these next