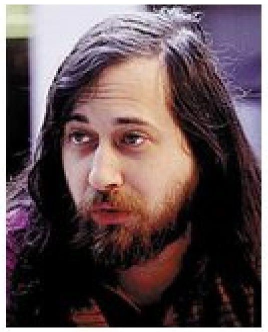
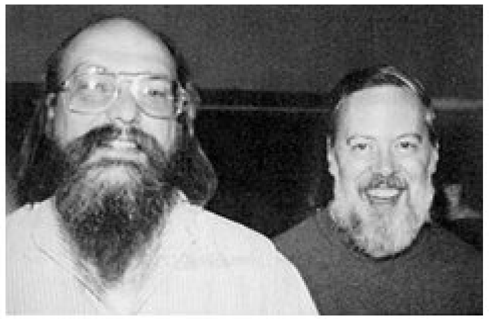
Basic programming in Bash

Linux environment

- UNIX is an operating system originally developed in AT&T's Bell labs in the 1970s
- AT&T had to provide the source code to anyone who asked
- GNU is a UNIX-based open source project started in the 1980s
- Linux was first released in 1991 and is considered (by some) to be a part of the GNU project

Torvalds, Linux


Stallman, GNU


Thompson & Ritchie, AT&T
(not shown: McIlroy and Ossanna)

Bash

- Bash is a command language interpreter
- It is a Shell, a user interface (command-line interface)
- Sophisticated execution of commands is possible through Bash scripts
- In bash, everything is a file
  - It can have Read (r), Write (w) and/or Execute (x) permissions

Simple Bash commands

`cd`	- change directory
`ls`	- list directory
`cat`	- concatenate and print file
`head`	- print beginning of the file
`tail`	- print end of the file
`wc`	- word count
`rm`	- remove
`mkdir`	- make directory
`man`	- show manual of a command (quit by pressing 'q')
`less`	- show file content
`pwd`	- show current directory

Motivation

- Basic programming is useful as it allows you to automate tasks
- MMseqs2 software suite allows creating tailored computational tools by combining its modules and workflows in Bash scripts

createdb

taxonomy

filterdb

search

createdb

search

filterdb

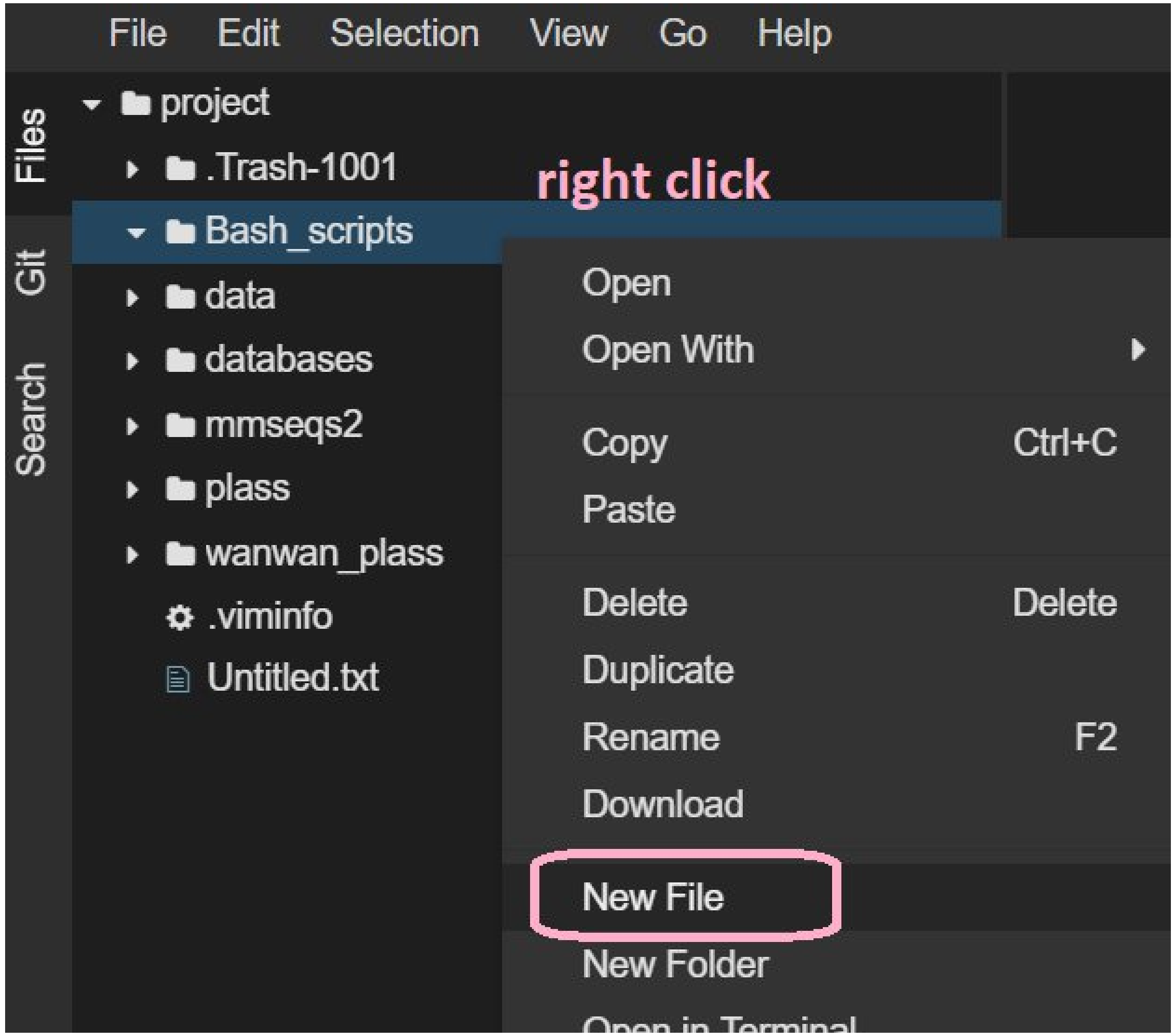
The script file

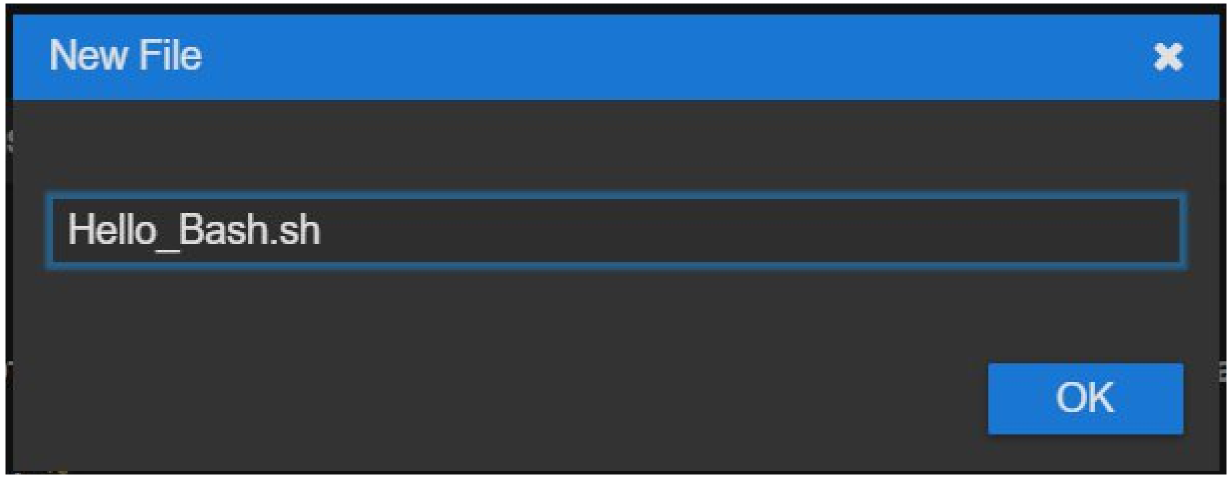
- The first line of a Bash script is usually:

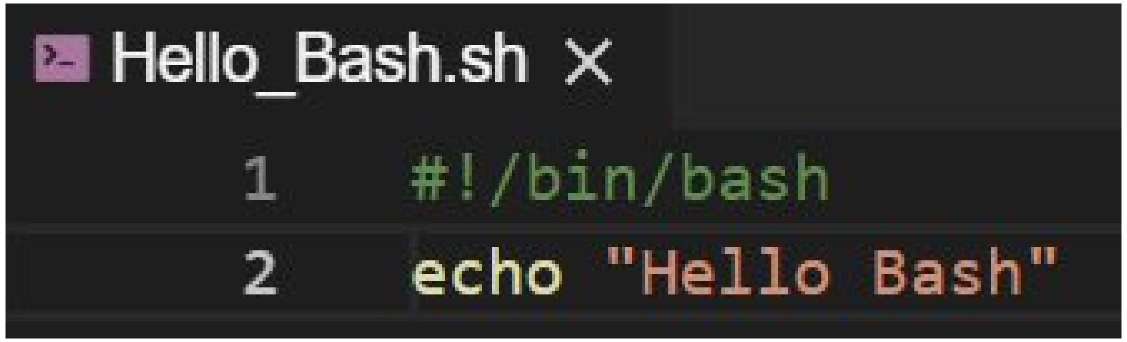
  ```
  #!/bin/bash
  ```

- This indicates this file is a Bash script
- Lines that start with ‘`#`’ are comments
- To print something we use ‘`echo`’
- A script is just a text file.
- **Under your home directory, create a directory called “Bash_scripts”**
- We will create Bash scripts there

Creating the Hello_Bash.sh script file







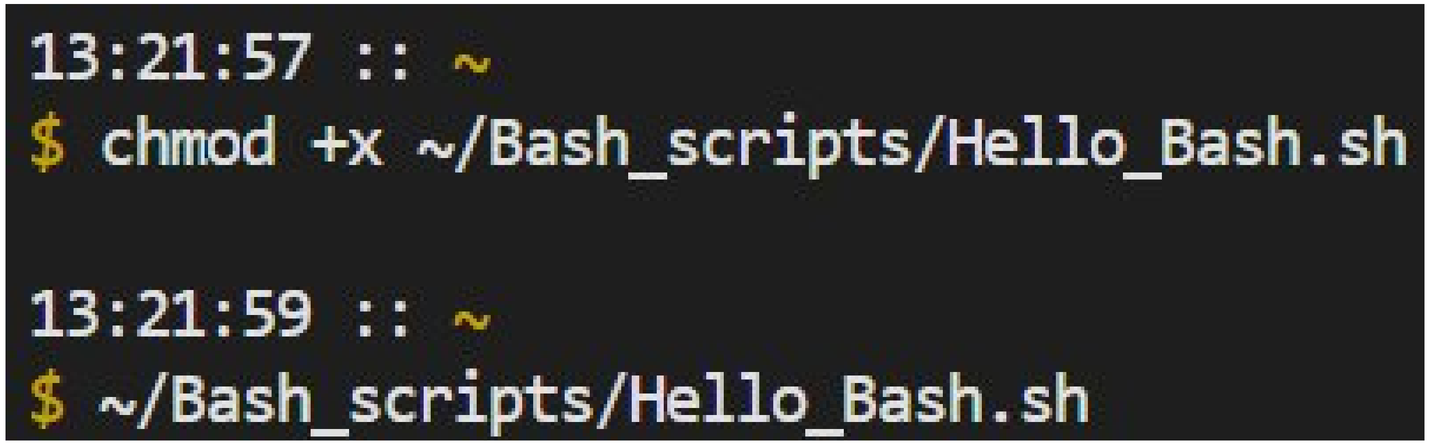
```
#!/bin/bash
echo "Hello Bash"
```

Running a Bash script

- You need to give your script execution permission:

```
chmod +x ~/Bash_scripts/Hello_Bash.sh
```

- Then you can run it from the terminal:



```
13:21:57 :: ~
$ chmod +x ~/Bash_scripts/Hello_Bash.sh

13:21:59 :: ~
$ ~/Bash_scripts/Hello_Bash.sh
```

"~" means your **home directory**
try:

```
echo $HOME
echo ~
cd ~
```

Hello_Bash.sh

Create a Hello_Bash.sh script and run it

Bash variables

- A variable stores a value
- There are no variable types in Bash
- Assignment of a value is done with “=“

```
#!/bin/bash
NAME="Eli"
NUMBER_OF_EYES=3
echo "Hello $NAME, you have $NUMBER_OF_EYES eyes"
```

- **Modify the Hello_Bash.sh script to have a variable and run it**

Arithmetic evaluation

- In order for bash to treat the variable as numeric we need to use brackets:

```
CORRECT_NUMBER_OF_EYES=$((NUMBER_OF_EYES - 1))
echo "Humans usually don't have more than
$CORRECT_NUMBER_OF_EYES eyes"
```

- **Create a Bash script with a variable AGE and assign it your age. Print the age you will be in one year**

Conditionals

- If/else structures allow us to execute commands only in certain cases

```
AGE=20
if [ "$AGE" -eq 20 ]; then
  echo "Wow, you are exactly 20!"
fi
```

- Comparison operators:

Description	Numeric	String
less than	-lt	<
greater than	-gt	>
equal	-eq	=
not equal	-ne	!=
less or equal	-le	
greater or equal	-ge	

User interaction

- This simple Bash script asks the user for their name and says hi:

```
#!/bin/bash
echo "Enter your name and press [ENTER]: "
read NAME
echo "Hi $NAME"
```

- **Create a script that asks for the user's age and serves beer only if the user is at least 18**

What does this code do?

```
echo "Enter a directory name and press [ENTER]: "
read DIR
if [ -d "$DIR" ]; then
    ls "$DIR"
else
     mkdir "$DIR"
fi
```

Repetitive execution of commands

- Often we would like to perform the same thing more than once:
  - Say hello to all students in the class (there are 22 of you!)
  - Make a copy of each file in a directory
  - Refine an MMseqs2 clustering…

- Bash loops allow us to do exactly that!

For loop

```
#!/bin/bash
START=1
END=22
for (( i=$START; i<=$END; i++ ))
do
    echo "$i. Hi, student!"
done
```

While loop

```
# continue from last slide
i=1
while [[ $i -le $END ]]
do
    echo "$i. Oh hi there, student!"
    ((i = i + 1))
done
```

Exercises

1. **Compute the sum of the first 40 natural numbers:**

   1+2+…

2. **Sum the numbers the user provides you until they provide a negative number**

   Can you tell how many numbers you summed?

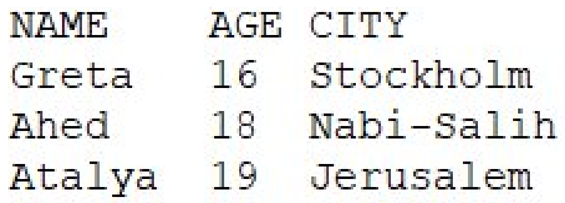
Text files: select columns

`cut` command let's you select columns from a text file

Flags:

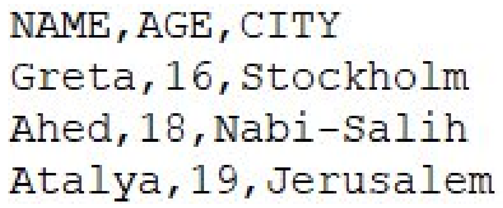
- `-f:` indicates columns to print (e.g.: 1,4-9,12-)
- `-d:` specifies column separator character (e.g.: ",")

tab separated



```
NAME    AGE CITY
Greta   16  Stockholm
Ahed    18  Nabi-Salih
Atalya  19  Jerusalem
```

comma separated



```
NAME,AGE,CITY
Greta,16,Stockholm
Ahed,18,Nabi-Salih
Atalya,19,Jerusalem
```

Redirect operator

> and >> redirects the Standard Output (stdout) to a file or elsewhere

- '>' creates and/or overwrites the file
- '>>' appends to the end of the file

Exercise: from the file '`molbio_2019.txt`' print the country of origin to a file called '`nationalities.txt`'

Pipe operator

We can easily transfer the output of one command to another using pipes

```
command1 | command2 | command3 ...
```

What do these commands do?

```
uniq nationalities.txt
sort nationalities.txt | uniq
```

More pipes

And these ones?

```
sort nationalities.txt | uniq | wc -l
sort nationalities.txt | uniq -c
sort nationalities.txt | uniq -c | sort -nrk1
```

Use the `man` command to find out what those flags mean

```
man sort
man uniq
man wc
```

`grep`

`grep <pattern> <file>` - extracts and prints all the lines that match a specific pattern or string in the files

`-c`: counts occurrences of the pattern

`-v`: print only the lines that DO NOT contain the pattern

`-i`: case insensitive flag

Exercises:

1. Count number of students from 'India'
2. Count number of students that are not from 'Germany'
3. How many people contain the the word 'an' in their names?

`grep`

`-E`: let's you use 'regular expressions'

What does this command do?

```
grep -E "^\w{5}\s" molbio_2019.txt
```

`grep` - Regular Expressions (regex)

```
grep -E "^\w{5}\s" molbio_2019.txt

'^'  : begin the line with this regex
'\w' : any letter
'{5}': exact n° of occurrences of last element
'\s' : any white space character
```

Exercise solutions

```
#!/bin/bash
echo "Hello Bash"
```

Exercise solutions

```
#!/bin/bash
AGE=99
AGE_NEXT_YEAR=$((AGE + 1))
echo "Next year you will be $AGE_NEXT_YEAR"
```

Exercise solutions

```
#!/bin/bash
echo "Enter your age and press [ENTER]: "
read USER_AGE
if [ $USER_AGE -ge 18 ]; then
    echo "Here is your beer"
fi
```

Exercise solutions

```
#!/bin/bash
START=1
END=40
SUM=0
for ((i=$START; i<=$END; i++)) do
    SUM=$((SUM+i))
done
echo "The result is $SUM"
```

Exercise solutions

```
#!/bin/bash
USER_NUMBER=0
NUM_NUMBERS=-1
SUM=0
while [[ $USER_NUMBER -ge 0 ]]
do
     SUM=$((SUM+USER_NUMBER))
     NUM_NUMBERS=$((NUM_NUMBERS+1))
     echo "Insert a new number [negative number to exit]:"
     read USER_NUMBER
done
echo "Final sum is $SUM and $NUM_NUMBERS numbers were summed"
```