High Performance Computing & Big Data Services

hpc.uni.lu

hpc@uni.lu

@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

Uni.lu HPC School 2021

PS1: Introduction to UNIX/Linux Shell and basic CLI

Uni.lu High Performance Computing (HPC) Team

T. Valette, A. Olloh, H. Cartiaux

University of Luxembourg (UL), Luxembourg

http://hpc.uni.lu

Latest versions available on Github:

UL HPC tutorials: https://github.com/ULHPC/tutorials

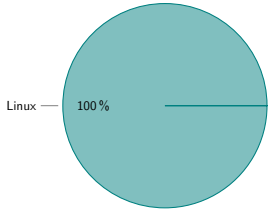
UL HPC School: hpc.uni.lu/education/hpcschool

PS1 tutorial sources: ulhpc-tutorials.rtfd.io/en/latest/linux-shell

Summary

Practical Session Objectives

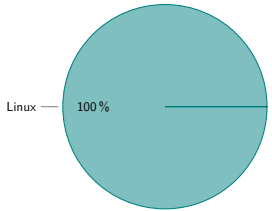
- HPC supercomputers are **exclusively** Linux-based
  - ↪ Note: Used to be Unix before
  - ↪ **better to become familiar with Linux environments**
    - ✓ interaction can be done from **ANY** OS
- Reasons:
  - ↪ stability and development flexibility



[Source : www.top500.org, Jun 2021]

Practical Session Objectives

- HPC supercomputers are **exclusively** Linux-based
  - ↪ Note: Used to be Unix before
  - ↪ **better to become familiar with Linux environments**
    - ✓ interaction can be done from **ANY** OS
- Reasons:
  - ↪ stability and development flexibility

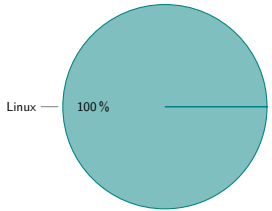


[Source : www.top500.org, Jun 2021]

⇒ How can I have a Unix shell on my operating system ?

Practical Session Objectives

- HPC supercomputers are **exclusively** Linux-based
  - ↪ Note: Used to be Unix before
  - ↪ **better to become familiar with Linux environments**
    - ✓ interaction can be done from **ANY** OS
- Reasons:
  - ↪ stability and development flexibility



[Source : www.top500.org, Jun 2021]

⇒ How can I have a Unix shell on my operating system ?

Objectives of the session (Part I)

- Install a Unix Shell.
- Open a new Unix shell.

Setup my environment

Hands-on: Pre-requisites Setup

▶ url ◀ | github | src

- **Linux**: Nothing to do
  - ↪ Already present if you're using CentOS, Ubuntu, RHEL, ArchLinux, RockyLinux...
  - ↪ Default shell available, Recommended terminal: Guake, Terminator...

Setup my environment

Hands-on: Pre-requisites Setup

▶ url ◀ | github | src

- **Linux**: Nothing to do
  - ↪ Already present if you're using CentOS, Ubuntu, RHEL, ArchLinux, RockyLinux...
  - ↪ Default shell available, Recommended terminal: Guake, Terminator...
- **Mac OS**: Nothing to do also,
  - ↪ in Finder, open `Applications` –> `Utilities` folder, then double-click `Terminal`
  - ↪ You probably want to install iTerm2

Setup my environment

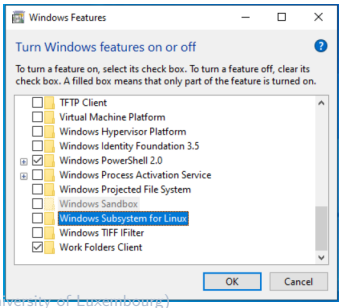
Hands-on: Pre-requisites Setup

▶ url ◀ | github | src

- **Linux**: Nothing to do
  - ↪ Already present if you're using CentOS, Ubuntu, RHEL, ArchLinux, RockyLinux...
  - ↪ Default shell available, Recommended terminal: Guake, Terminator...
- **Mac OS**: Nothing to do also,
  - ↪ in Finder, open `Applications` –> `Utilities` folder, then double-click `Terminal`
  - ↪ You probably want to install iTerm2
- **Windows**: We recommend you to install
  - ↪ Windows Subsystem for Linux (WSL)
  - ↪ Ubuntu over WSL
  - ↪ Windows Terminal optional.

Setup my environment: Windows

- **Windows**
  - ↪ WSL Install
    - ✓ In the start Menu: select “Turn Windows features on or off”
    - ✓ Tick “Windows Subsystem for Linux”
    - ✓ Reboot to complete the install



Setup my environment: Windows

- **Windows**
  - ↪ In the start Menu: select "Developers Settings"
    - ✓ Turn on Developer Mode
  - ↪ Install Ubuntu over WSL
    - ✓ Now you can **Install Ubuntu within the Microsoft Store**
    - ✓ Click on "Launch" or from the start Menu, select Ubuntu 20.04 LTS and press Enter.
    - ✓ You'll get a prompt to create your **username** and **password**

Summary

Introducing the Unix Shell

Questions

- What is a command shell and why would I use one ?

Objectives

- Explain how the shell relates to the keyboard, the screen, the operating system and users' programs.
- Explain when and why command-line interfaces should be used instead of graphical interfaces.

Background

- How do we interact with personal computers ?
  - ↪ With a GUI (Graphical User Interface)
  - ↪ We give instruction by clicking a mouse and using menu-driven interactions.
- Why ?
  - ↪ GUI makes things intuitive to learn
- Imagine the following task and how would you solve it ?
  - ↪ you have to copy the third line of one thousand text files in one thousand different directories and paste it into a single file.

Introducing the Unix Shell

Definition: shell

- **Shell**: program allowing to send commands to the computer and receive output.
  ↪ It is also referred to as the **terminal** or **command line interface (CLI)**.

- **Type of Unix Shell**: `zsh`, C shell (`csh`), `sh`, `ksh`, `Bash`.
  ↪ The most popular is Bash (Bourne Again Shell)
- **Why should I use a shell?**
  ↪ The **easiest way** to interact with remote [Linux] machines **and** supercomputers.
  ✓ Many tools only have command-line interfaces
  ↪ Allows you to **combine** existing tools in powerful ways (pipe etc.) to create new tools of your own with little or no programming.

Introducing the Unix Shell

Unix Shell from Software Carpentry

https://swcarpentry.github.io/shell-novice/

Hands-on: Introducing the Shell (SW Carpentry)

▶ url

- Opening a terminal
  - ↪ $ (generally) denotes the **prompt** and is **NOT** part of the command
  - ↪ sometimes, $> is also used in some documentation
- Your very first command: `ls`
- Your (first but not last) commands not found

Navigating Files and Directories

Questions

- How can I move around on my computer ?
- How can I see what files and directories I have ?
- How can I specify the location of a file or directory on my computer ?

Objectives

- Explain the similarities and differences between a file and a directory.
- Translate an absolute path into a relative path and vice versa.
- Construct absolute and relative paths that identify specific files and directories.
- Use options and arguments to change the behaviour of a shell command.
- Demonstrate the use of tab completion and explain its advantages.

Home Directory Variation

- Who am I

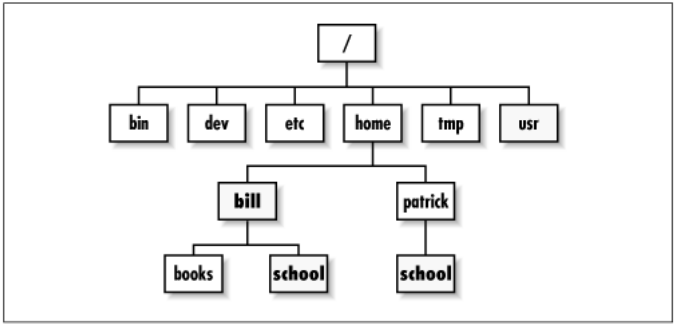
```
$ whoami
aolloh
```

- Find out where I am: `pwd`
  - ↪ stands for "print working directory", it returns the user's home directory.

```
$ pwd
/home/aolloh
```

- On Windows, it will be similar to `C:\Documents and Settings\Users\aolloh`

Understand Linux File Tree



- At the top is the **root directory** that holds everything else.
  - ↪ We refer to it using a slash character, /

Understand Linux File Tree

- Notice that there are two meanings for the / character.
  - ↪ When it appears at the **front** of the file or directory name, it refers to the root directory name.
  - ↪ When it apears **inside** a path, it's just a separator.
- Explain some directories
  - ↪ `bin` : contains built-in programs
  - ↪ `home` : users' personal directories
  - ↪ `tmp` : contains tempory files that don't need to be stored long-term
  - ↪ `boot` : boot files system
  - ↪ `var` : contains variables
  - ↪ `dev` : device files
  - ↪ `etc` : configuration files

List the contents of my own filesystem

- Prints the names of the files and directories: `ls`
  - ↪ stands for "listing"

```
$ ls
Applications Documents    Library      Music        Public
Desktop      Downloads    Movies       Pictures
```

- Use option
  - ↪ a trailing / indicates a directory
  - ↪ @ indicates a link
  - ↪ * indicates an executable

```
$ ls -F
Applications/ Documents/    Library/      Music/        Public/
Desktop/      Downloads/    Movies/       Pictures/
```

List the contents of my own filesystem

- Clear your terminal

```
$ clear
```

- Getting help

```
$ ls --help
$ man ls
```

- Unsupported command-line options

```
$ ls -j
ls: invalid option -- 'j'
Try 'ls --help' for more information.
```

Let's practise

Your Turn!

Hands-on: Navigating Files and Directories (SW Carpentry)

▶ url

- Home Directory Variation
- Understanding Slashes
- Getting Help
- Unsupported command-line options
- The `man` command

```
# What would output this command?
$ man man
```

Explore other directories

- List something other than your current directory

```
$> ls [option] [path]
```

```
$ ls -F Desktop
$ ls -F Desktop/shell-lesson-data
```

- Move from your home to other directory: cd
  - ↪ stand for "change directory"
  - ↪ without an argument it will return you to your home directory.

```
$> cd [path]
```

```
$ cd Desktop/shell-lesson-data
```

Explore other directories

- Leave a directory and go into its parent directory
  - ↪ the simpliest way: `cd ..`
  - ↪ `..` means the **parent** of the current directory.

```
$ cd Desktop/shell-lesson-data/data
$ pwd
/home/aolloh/Desktop/shell-lesson-data/data
$ cd ..
$ pwd
/home/aolloh/Desktop/shell-lesson-data
$ ls -a -F
./   .bash_profile  data/       north-pacific-gyre/  pizza.cfg  thesis/
../  creatures/     molecules/  notes.txt            solar.pdf  writing/
```

Explore other directories

- Hidden files
  - ↪ usually contains shell configuration settings. Ex: `.bash_profile`
  - ↪ prefix by `.`
  - ↪ a standard `ls` command doesn't print them

Explore other directories

- Relative path
  - ↪ When you use a **relative path** with a command like `ls` or `cd`, it tries to find that location from where we are, rather than from the root of the file system.

```
$ cd Desktop/shell-lesson-data/data
```

- Absolute path
  - ↪ However, it is possible to specify **the absolute path** to a directory by including its entire path from the root directory, which is indicated by a leading slash.

```
$ cd /home/aolloh/Desktop/shell-lesson-data/data
```

Explore other directories

- Shorcuts
  - ↪ ~ (tilde) character at the start of a path to mean **the current user's home directory**
  - ↪ - (dash) character cd will translate - into **the previous directory**
  - ↪ The difference between `cd ..` and `cd -` is that the former brings you **up**, while the latter brings you **back**

```
$ cd ~/Desktop/shell-lesson-data
$ cd creatures
$ cd -
```

Let's practise

Your Turn!

Hands-on: Exploring Other Directories (SW Carpentry)

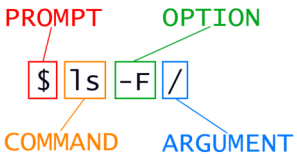
▶ url

- `cd` and `pwd`
- Hidden files
- the `~` shortcut and `cd -` command
- Absolute vs Relative Paths
- Relative Path Resolution
- `ls` Reading Comprehension

What you learned so far...

- The file system is responsible for managing information on the disk.
- Information is stored in files, which are stored in directories (folders).

- **Command anatomy**
  - ↪ Most commands take *options (flags)* that begin with a `-`.
  - ↪ **Ex**: `ls -l` or `ls -a` or `ls -la`



Command	Description
`cd [path]`	changes the current working directory.
`ls [path]`	prints a listing of a specific file or directory; `ls` on its own lists the current working directory.
`pwd`	prints the user's current working directory.

Navigating with Files and Directories - Key Points

- A **relative** path specifies a location starting from the current location.
- An **absolute** path specifies a location from the root of the file system.
- Special directories
  - ↪ / is the **root** directory of the whole file system
  - ↪ ~ is the **home** directory
  - ↪ .. means the directory above (**parent** of) the current one
  - ↪ . means 'the current directory'
- **Directory names** in a path are **separated** with / on Unix, but \ on Windows.
  - ↪ Ex (Unix): `/Users/toto`
  - ↪ Ex (Windows): `C:\Users\toto`

Working with files and directories

Questions

- How can I create, copy, and delete files and directories?
- How can I edit files?

Objectives

- Create a directory hierarchy.
- Create files using an editor or by copying and renaming existing files.
- Delete, copy and move specified files and/or directories.

Creating files & directories

- Create files: `touch <filename>`
  - ↪ It will "touch" / create an empty file (exists but no content)
- Create directories: `mkdir <dirname>/`
  - ↪ stands for "make directory", it'll create an empty directory
  - ↪ The -p option allows mkdir to create a directory with nested subdirectories in a single operation

Files and directories naming conventions

- Several common rules which should help you name your files and directories
  - ↪ do not use white spaces
  - ↪ do not use leading dash
  - ↪ stick to alphanumerical characters and dot, underscore, dash characters ([a-ZA-Z._-])

Warning!

- If you do not follow these recommandations, you will have to surround your name files with quotes (“file name”, “$superdir”, . . .)

Files and directories naming conventions

- Several common rules which should help you name your files and directories
  - ↪ do not use white spaces
  - ↪ do not use leading dash
  - ↪ stick to alphanumerical characters and dot, underscore, dash characters ([a-ZA-Z._-])

Warning!

- If you do not follow these recommandations, you will have to surround your name files with quotes (“file name”, “$superdir”, . . .)

- File extensions
  - ↪ You can and should use file extensions
  - ↪ Linux/UNIX doesn’t require any file extension, but if you don’t use them . . . you won’t be able to differentiate a binary file from an image

Copy & Move files and directories

- `cp` command stands for `copy`
  - ↪ used to copy a file
  - ↪ or a directory and its content with the `-r` option (recursive copying)
- `mv` command stands for `move`
  - ↪ used to move a file or a directory from a location to another
  - ↪ is also used to rename a file or a directory

Delete files and directories

- `rm` command stands for `remove`
  - ↪ used to remove a file
  - ↪ or a directory and its content with the `-r` option (recursive removing)

```
$> rm [-r] [-i] path
```

Warning!

- Please be careful when using `rm`
  - ↪ there'll be no confirmation (except if you use `-i` option for interactive deletion)
  - ↪ this operation is not reversible, once a file or directory is deleted, you can't restore it
  - ↪ even if tools can't try to restore your files after a deletion, it's not guaranteed
- Issuing a `cp` or a `mv` with an already existing destination path will act as a deletion then recreation of the destination
- Backup your data to avoid losing your data in case of mistakes

Wildcards

- Wildcards is a way to access multiple files or directories at once.
  - ↪ You can consider it as a joker in a file name and will try to match every file/directories names in your current location
- * matches zero or more characters

```
tvalette@laptop ~/tutorial $ ls -F
aion1/  aion2/  aion3/  aion4/  aion5/  iris1/  iris2/  iris3/  iris4/  iris5/
tvalette@laptop ~/tutorial $ ls -F aion* # aion* will be expanded into
                                         # "aion1 aion2 aion3 aion4 aion5"
aion1:
aion2:
aion3:
aion4:
aion5:
```

Wildcards

- Wildcards is a way to access multiple files or directories at once.
  - ↪ You can consider it as a joker in a file name and will try to match every file/directories names in your current location
- ? matches exactly one character

```
tvalette@laptop ~/tutorial $ ls -F
aion1/  aion2/  aion33/  aion44/  aion55/  iris1/  iris22/  iris33/  iris44/  iris55/
tvalette@laptop ~/tutorial $ ls -F aion? # aion? will be expanded into "aion1 aion2"
aion1:
aion2:
```

Text editors

- `nano` is one of the simplest **text** editor
  - ↪ only text possible
- `vim` is a more complex and powerful **text** editor
  - ↪ it can be use a complete IDE and has many very interesting features that can significantly speed up your development workflow
- Religious alternative: `emacs`
- GUI alternatives (**NOT** meant to be used remotely easily)
  - ↪ Atom, Sublime Text...

Let's practise

Your Turn!

Hands-on: Working With Files and Directories (SW Carpentry)

▶ url

- Creating directories
- Create a text file
  - ↪ play with various editors (`nano` etc.)
  - ↪ **Q**: how to exit `vim`
- Moving files and directories
- Copying files and directories
- Removing files and directories
- Operations with multiple files and directories
- Using wildcards for accessing multiple files at once

Working with files and directories – Key points

Command	Description
`cp [old] [new]`	copies a file.
`mkdir [path]`	creates a new directory.
`mv [old] [new]`	moves (renames) a file or directory.
`rm [path]`	removes (deletes) a file.

- The shell **does not** have a trash bin:
  - ↪ **once something is deleted, it's really gone!!!**.
- Most files' names are on the form: `something.extension`
  - ↪ Normally used to indicate the type of data in the file
  - ↪ **extension is not** required, and doesn't guarantee anything.

Working with files and directories – Key points

- Depending on the type of work you do, you may need a **more powerful** editor than Nano.
  - ↪ `vim`, `emacs` etc.
- **Widcards**
  - ↪ `*` matches zero or more characters in a filename, so `*.txt` matches all files ending in `.txt`.
  - ↪ `?` matches any single character in a filename, so `?.txt` matches `a.txt` but not `any.txt`.

Summary

Pipes and Filters

Questions

- How can I combine existing commands to do new things?

Objectives

- Redirect a command's output to a file.
- Process a file instead of keyboard input using redirection.
- Construct command pipelines with two or more stages.
- Explain what usually happens if a program or pipeline isn't given any input to process.
- Explain Unix's 'small pieces, loosely joined' philosophy.

Filtering text

Command	Description
`cat <file>`	Print file text without any filtering
`sort <file>`	Sorting a file
`head <file>`	Getting first lines
`tail <file>`	Getting first lines
`uniq <file>`	Ensure each line are different/unique lines
`wc [-l] <file>`	Counting words [Lines with -l] in file

- Other powerful tools you may want to know: **sed** and **awk**
  - ↪ **sed**, for instance, to replace every match of the pattern into specified string

```
# replace every match of the pattern into specified string
sed 's/<pattern>/<string>/i' <file>
# print every match of the specified pattern:
awk '/<pattern>/ {print $0}' <file>
```

Capturing output

- > is used to use output from a command and store it in a file
  ↪ be careful, using > with an existing file will overwrite the destination

Capturing output

- > is used to use output from a command and store it in a file
  ↪ be careful, using > with an existing file will overwrite the destination
- alternatively, you can use >> if you want to append your output to a file without overwriting it

Passing output to another command

- > and >> are very powerful, but they have a major drawback if you want to chain multiple tools:
  - ↪ you need to create a file for each text: `program 1 -> file -> program 2 -> file -> program 3 -> ...`
- in order to get rid of these intermediary files, you will have to use pipes : |
  - ↪ on Azerty keyboards: `Alt+6`
  - ↪ on Qwerty keyboards: `Shift+/`, but can be different depending on your keyboard layout

Let's practise

Your Turn!

Hands-on: Pipes and Filters (SW Carpentry)

▶ url

- Capturing output from commands
- Filtering output
- Redirecting output
- Passing output to another command
- Combining multiple commands
- Reviewing Tools designed to work together

Pipes and Filters – Key points

Command	Description
`wc`	counts lines, words, and characters in its inputs.
`cat`	displays the contents of its inputs.
`sort`	sorts its inputs.
`head`	displays the first 10 lines of its input.
`tail`	displays the last 10 lines of its input.
`command > [file]`	redirects a command's output to a file (overwriting any existing content).
`command >> [file]`	appends a command's output to a file.

- `<cmd1> | <cmd2>` is a **pipeline**:
  - ↪ **output** of the first command `<cmd1>` is used as **input** to the second one `<cmd2>`
- The best way to use the shell is to use pipes to combine simple single-purpose programs (filters).

Loops

Questions

- How can I perform the same actions on many different files?

Objectives

- Write a loop that applies one *or more* commands separately to each file in a set of files.
- Trace the values taken on by a loop variable during execution of the loop.
- Explain the difference between a variable's name and its value.
- Explain why spaces and some punctuation characters shouldn't be used in file names.
- Demonstrate how to see what commands have recently been executed.
- Re-run recently executed commands without retyping them.

For loop structure

- You can use for loops in order to repeat same actions multiple times across a list

```
for thing in list_of_things; do
    operation_using $thing    # Indentation within the loop is
                              # not required, but aids legibility
done
```

- `list_of_things` can be various lists such as
  - ↪ a list of files: `file1.txt file2.txt file3.txt [...]`
  - ↪ selection of files with a wildcard: `file*.txt` or `*.csv`
  - ↪ a list of string or number: `1 2 3 5 [...]` or `chaos iris aion [...]`
  - ↪ a range: `{0..15}` (from 0 to 15)

For loop variable

- `thing` will be a variable in your for block
  - ↪ `$thing` is usable inside your block and will contain every item of your `list_of_things` over iterations
  - ↪ for instance, with the following list: `{1..3}`, then `$thing` will contain 1, then 2, then 3
- Please **stick to naming convention presented sooner for your variable names**
  - ↪ using non-alphanumerical characters (or `[._-]`) will force you to surround your variables with quotes, and you'll have more problem than anything
    - ✓ `$my_variable` is a good name
    - ✓ `$\$my_\^variable` is **not** a good name

Using variables in shell script

- When using variables in shell scripts, you are encouraged to use the syntax `${variablename}`

Loops construction recommandations

- **When writing your for loop, your prompt will change**
  - ↪ This tells you if you are in a block or not
    - ✓ **if you see $**: you can type a command as usual, you are not in a block
    - ✓ **if you see >**: you are in a block and commands you'll issue will be repeated in the loop

Loops construction recommandations

- **When writing your for loop, your prompt will change**
  - ↪ This tells you if you are in a block or not
    - ✓ **if you see $**: you can type a command as usual, you are not in a block
    - ✓ **if you see >**: you are in a block and commands you'll issue will be repeated in the loop

- **Use dry-run** when writing loops to be sure is behaving how you expect
  - ↪ **add `echo` before each of your commands**
    - ✓ you'll print your full command with evaluated variables, yet without executing them

```
for thing in list_of_things; do
    echo operation_using $thing
done
```

Loops and shell features

- Interesting facts, as you can reproduce anything on a shell inside a for loop, you can:
  - ↪ use outputs redirections (>)
  - ↪ use pipelines (|)
  - ↪ use **nested loops** (other for loops inside a for loop)

```
$ for cluster in iris aion; do
>   for node in {1..100}; do
>       echo "${cluster}-${node}" >> ${cluster}-nodelist.txt
>   done
> done
$ head -n 2 iris-nodelist.txt
iris-1
iris-2
$ tail -n 2 aion-nodelist.txt
aion-99
aion-100
```

Facilitate your shell journey

- `history` and up/down arrow to browse your history
- **Very important**: `man <command>` to get documentation of a command

https://blog.ssdnodes.com/blog/cheatsheet-bash-shortcuts/

Productivity-boosting Bash shortcuts

Command	Description
`Ctrl+a`	to go to the beginning of the line
`Ctrl+e`	to go to the end of the line
`Ctrl+r`	to search in history commands (`Ctrl+r` again to keep searching)
`Ctrl+k`	Cut the **line** **after** the cursor to the clipboard
`Ctrl+w`	Cut the **word** **before** the cursor to the clipboard.
`!!`	to rerun last command
`!$`	to get last word from last command
...	...

Let's practise

Your Turn!

Hands-on: Loops (SW Carpentry)

▶ url

- Follow the Prompt
- Playing with Variables in Loops
- Limiting Sets of Files
- **Avoid** but comply with Spaces in Names

Loops – Key points

- A `for` loop repeats commands once for every thing in a list.
- Every `for` loop needs a variable to refer to the thing it is currently operating on.
- Use `$name` to expand a variable (i.e., get its value). `${name}` can also be used.
- Do not use spaces, quotes, or wildcard characters such as '*' or '?' in filenames, as it complicates variable expansion.
- Give files consistent names that are easy to match with wildcard patterns to make it easy to select them for looping.
- Use the up-arrow key to scroll up through previous commands to edit and repeat them.
- Use `Ctrl+R` to search through the previously entered commands.
- Use `history` to display recent commands.

Summary

Shell Scripts

Questions

- How can I save and re-use commands?

Objectives

- Write a shell script that runs a command or series of commands for a fixed set of files.
- Run a shell script from the command line.
- Write a shell script that operates on a set of files defined by the user on the command line.
- Create pipelines that include shell scripts you, and others, have written.

Key points

- Save commands in files (usually called shell scripts) for re-use.
- `bash [filename]` runs the commands saved in a file.
- `$@` refers to all of a shell script's command-line arguments.
- `$1`, `$2`, etc., refer to the first command-line argument, the second command-line argument, etc.
- Place variables in quotes if the values might have spaces in them.
- Letting users decide what files to process is more flexible and more consistent with built-in Unix commands.

Explain Shell Online

https://explainshell.com

Shell Lint / Syntax Checker Online

https://shellcheck.net

Let's practise

Your Turn!

Hands-on: Shell Scripts (SW Carpentry) ▶ url

- Creating script file
- Double-Quotes Around Arguments
- List Unique Species

Summary

Finding things

Questions

- How can I find files?
- How can I find things in files?

Objectives

- Use `grep` to select lines from text files that match simple patterns.
- Use `find` to find files and directories whose names match simple patterns.
- Use the output of one command as the command-line argument(s) to another command.
- Explain what is meant by ‘text’ and ‘binary’ files, and why many common tools don’t handle the latter well.

Key points

- find finds files with specific properties that match patterns.
- `grep` selects lines in files that match patterns.
- `--help` is an option supported by many bash commands to display more information on how to use these commands or programs.
- `man [command]` displays the manual page for a given command.
- `$([command])` inserts a command's output in place.

Let's practise

Your Turn!

Hands-on: Finding Things (SW Carpentry)

▶ url

- Discovering `grep`
- To filter or **not** to filter
- Wildcard magics
- Ex: Tracking a Species
- Ex: Little ~~Wo~~men
- Matching and Subtracting

Summary

Thank you for your attention...

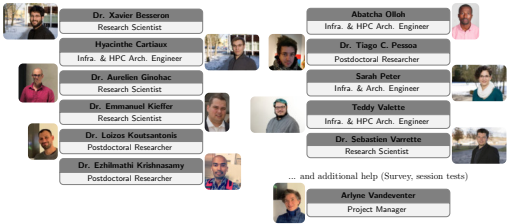
Questions?

ulhpc-tutorials.rtfd.io/en/latest/linux-shell

High Performance Computing @ Uni.lu

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

Uni.lu HPC School 2021 Contributors



hpc.uni.lu