BASH/SHELL WORKSHOP

For Data Science

PART 1

How the course is organized?

- Bash
  - basics, the environment, other tools available to bash environment, practical tips
  - basic bash commands
  - advanced bash commands
  - bash scripting syntax
  - creating & running scripts
- Tutorial: Makefiles, interplaying with bash scripts, and introduction to project
- Tutorial: Manipulating and managing Github tasks in BASH environment.
- Tutorial: Data mining and wrangling in BASH environment
- Use bash on processing real-world data sets

Access to a shell linux environment

Instructions

1. Install a linux OS, recommend Ubuntu, on a virtual machine on windows, such as virtualbox (or vmware)
   - https://www.wikihow.com/Install-Ubuntu-on-VirtualBox
   - This works well for those who have older Windows versions
2. It is recommended to install WSL (windows subsystem for linux) on Windows 10. Allows you to install Ubuntu on windows, in a limited way, but enough to give you access to bash shell right in your Windows environment.
3. Create AWS account – Free tier, Create EC2 instance, SSH (Possible with azure too)
4. Perhaps you can get shell access to some machines on ULB network? (Your system administrators may know?)
5. Optional – GIT BASH, Atom editor (very limited features, not recommended)

WSL 2 set up instructions

- If you are running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11), you can check this by running 'winver' in your Windows Cmd prompt, then you can install wsl by simply typing 'wsl --install' in your Windows Cmd prompt
  - Or, see https://docs.microsoft.com/en-us/windows/wsl/install
- If not (running an older Windows 10), follow the process in https://docs.microsoft.com/en-us/windows/wsl/install-manual
- The end result is, you'll see a bash terminal program in your start-menu, 'search for Ubuntu' to locate it
- At this point, it is **strongly recommended** to install 'Windows Terminal' as it offers a much smoother experience when working with default bash terminal on windows.
  - Install from https://www.microsoft.com/en-us/p/windows-terminal/9n0dx20hk701#activetab=pivot:overviewtab

Text editors

Text Based

- Basic: okay for occasional use, simple to use
  - Nano
- Advanced: can be very productive environments for seasoned programmers who write a lot of code
  - Vim
  - EMACS

GUI

- ATOM
- Brackets
- Sublime
- ..

Basics

- All commands you type on a bash terminal prompt are little programs that are stored inside your computer
  - they are either bash built-in commands
  - or refer to tools installed in your linux environment, and callable from command line, at bash prompt
- General structure of a command
  - bash-prompt $ commandname OPTIONS INPUTS
- Examples
  - $ echo hello
  - $ cal
  - $ ls
  - $ mkdir directory_hello
- Which of these are ‘bash built-in commands’ and which are ‘other tools’?

General behavior of commands:

- What happens you type a command at the bash prompt
  - $ ls
    - a command to list files in the current directory
  - bash environment will check if it is a built-in command, if yes, it does that job
  - if not, it will try to ‘search’ the system if there’s a tool available by that name, and runs it if it finds it.
- How does it search? How does it know where to look?
- Then brings us to bash environment
  - a variable called PATH defines all the places (directories) bash should search to find the binary/executable tool called ‘ls’
  - try running ‘echo $PATH’?
- Manual pages
  - ‘man ls’
    - how to exit from manual page? press ‘q’
  - ‘man commandname’ -- not always, but most commands have a manual page
  - when they don’t
  - most commands print a small ‘Usage’ / ‘Help’ message when you pass the option ‘-h’ or ‘--help’
    - e.g., ‘ls -h’ or ls --help
- So what’s the story with ls? was it a bash built in, or another tool?
  - $ which ls

    may print ‘/usr/bin/ls’ or ‘/bin/ls’ -- depending on your Linux Distribution
  - so we know it is not a bash built-in

Standard input and outputs

- cat 1> output.txt
  Welcome to ULB!
- cat 1>> output.txt #Double greater sign (>>) than APPENDS to the file
- Standard error: cat –k ulb 2> error.txt
- Redirecting output and error simultaneously:
- cat 1> output.txt 2> error.txt
- Standard input: cat 0> input.txt
- Reading from standard input: cat < input.txt
- Try: cat 0< input.txt 1> test.txt
- Try: command tty in another window and redirect input to a new directory.

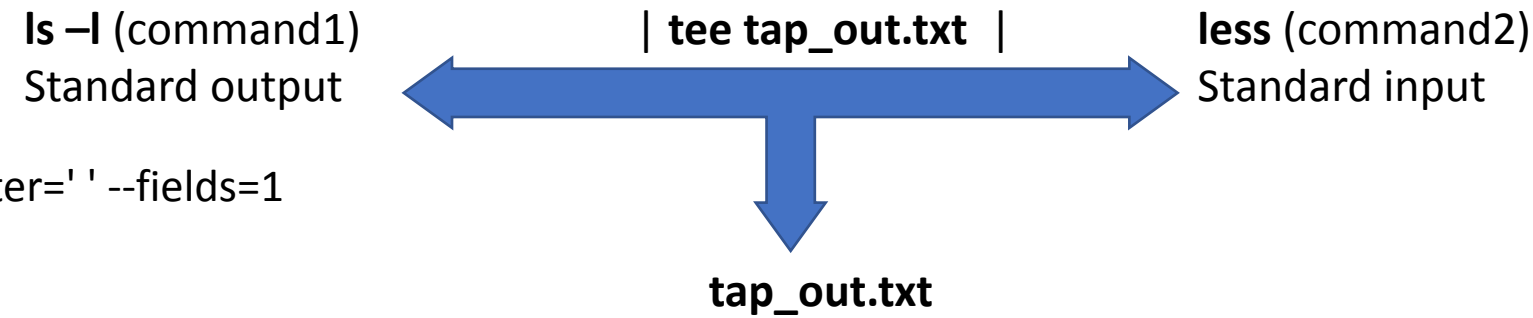
A preview of some things you can do at the bash prompt

Approach 1

- date > date.txt
- cut < date.txt --delimiter=' ' --fields=1

Approach 2

- date | cut --delimiter=' ' --fields=1
- date | cut --delimiter=' ' --fields=1 > vandag.txt
- What if you want to see what's going between the pipes before the output stream gets filtered by the next step in the pipeline?
  - command: tee



- date | tee vandag.txt | cut --delimiter=' ' --fields=1
- cat vandag.txt
  - to see what is in this file
  - another way to peek at what's in a file (a text file)
    - less vandag.txt

Creating File and Folder structure

- Folders and filenames are case sensitive in linux
- All files and folders in linux are arranged in a tree fashion, and are ‘rooted’ at a special directory named / -- yes, a slash
- Absolute paths to *all* files are specified starting from /
    - e.g., /home/sk, or /usr/bin/ls
        - / has a directory called usr, usr has a directory called bin, and bin has a file called ls
    - Try the following commands
        - file /usr/bin/ls
        - file /usr/bin
        - what is file?
            - man file
        - pwd # should print your present working directory

Creating directories

- mkdir hellodir1
- mkdir hellodir3 hellodir4
- mkdir -p project1/Announcements/ references/paper1
    - this creates project1, and a subdirectory named Announcements, in it, etc.,

Project structure: Brace Expansion

- Lets say you are working on a 2 year project and every month you need write/(or-generate!) 100 memo files to keep track of the project because that's the way administration people like it: creating all these files/folders can be tedious
- You can either do it with a bash script (we shall see later), you can imagine for-loops being involved….
- Or, there are some nifty features in bash that help you do it more compactly
- Try
  - echo {a,b,c}_{1,2,3}
  - You will see 'a_1 a_2 a_3 b_1 b_2 b_3 c_1 c_2 c_3' printed; now try echo {a,b}.{1,2,3,4}
  - Guess what is happening? -- something like a cartesian product (of two sets)
- mkdir project
- cd project
- mkdir {jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec}_{2021,2022}
- touch {jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec}_{2021,2022}/output{1..50}
- NOTE: bash has a number of such features, you don't have to know them all, acquire them gradually

Customising bash environment: custom commands

You may want your bash environment to

- remember the commands you type
- set-up your search paths (remember the $PATH variable?)
  - You may have your tools (your project/circumstance specific) in non-standard places
    - what are standard places → the output of 'echo $PATH' is the list of all standard locations
- set-up aliases to some of the frequently used long commands you use
- customize your bash-prompt to be more useful than it is
- ….do many more such things

To do that, we modify a file called $HOME/.bashrc

- This file, as it is, is bash script, and this script is run everytime you open a new bash shell

BASH Environment

- Try the following
  - $ set > $HOME/my_bash_env_on_`date +%Y_%m_%d`
  - less $HOME/my_bash_env_on_<press-tab>
  - Everything you see here is a bash command, and in total, this defines your bash environment! Scroll around and observe what all is in here.
- If you update $HOME/.bashrc, and open a new terminal, and redo the above step, your new updates will be reflected in there
- Examples of useful things to add to your $HOME/.bashrc? Coming soon.
  - There are other files, such as $HOME/.bash_aliases, to store command aliases such as below, however, you can put everything, in general, in $HOME/.bashrc
    - alias getdates="date | tee /tmp/fulldate.txt | cut --delimiter=' ' --fields=1 | xargs echo hello"