ABOUT THIS PAGE

This page contains some information you will find useful when working on this Linux cluster. This page can be shown with **showhelp** or printed as a pdf found at */home/hwalinga/bin/showhelp.pdf* or */data1/hwalinga/bin/showhelp.pdf*. If you view this on the command line with **showhelp**, you can quit this program with q (or :q). This will works for most terminal programs, by the way.

SMALL INTRODUCTION TO LINUX

OPERATING SYSTEM

Linux (or GNU+Linux) is an Operating System (OS), but you are currently only interacting with this OS by a terminal (or command line). The shell you are currently running is called **bash** (most popular by far). While **bash** is a complete programming language, you rarely use most of its language features. Most of the time you use external programs called from bash, but at least you know that something as *bash for loops* exists.

CALLING COMMANDS

The first word of what you type is the external command you are calling. Either you specify the full path, or bash can automatically find it in the *$PATH* variable. Using *echo $PATH* you can find all the directories that are in the *$PATH* variable (colon (:) separate). Using **which command** you can find the place **bash** found the **command**.

YOUR ENVIRONMENT

The *$PATH* variable is part of your environment. The environment can be different for every user and is set up by your *.bashrc* file found in your home directory (*/home/username/.bashrc*). This file will always be loaded at start-up and changes here make the changes persistent, but specific temporary changes can always be done during a session (some tools require this).

ALIASES

Another part of your environment are aliases. Aliases are a quick way to execute a complex command with a single word. I defined some aliases for your convenience and you can define your own in your local *bashrc* (*/home/username/.bashrc*). To find out what an alias defines you can call **type -a command**. For your convenience I aliased **which** to this (**alias which="type -a"**) so that **which** will show both aliases and commands in the path and also show all, not just the first (as is default by which).

AUTOCOMPLETE

You can autocomplete a lot of stuff with TAB. Commands, filenames, and some more stuff. Next to being useful, it is also a good indication that what you type is correct.

CONNECTING TO WEBDATA/BULK

Webdata or bulk drive can be accessed with the sftp protocol using the **sshfs** command, but I created a few aliases that will help you to access it quicker.

The idea is that the bulk drive is mounted to a folder in the file system, so that all files can be accessed and moved to, just as if the bulk drive is part of the normal file system.

mountbulk
: Mounting the bulk drive to */data1/bulk/username/shared/*. Some protocols are set-up so that the mounting is stable and consistent after logging out and logging in.

unmountbulk
: Unmount the bulk drive, because sometimes the connection has problems.

autobulk
: Unmount and than mounting it again.

forceautomountbulk
If there are really issues, this command will stop the sshfs (with force) and tries to reconnect.

The same commands exists for your home, by appending home to these commands. However you might check if these are the connected to the right path. (Assuming staff-homes and first name of your surname is the second letter of your NETID.)

Despite that a connection to the bulk should persist, in practise this rarely works. Allthough this problem has been reported, IT thinks it can be solved, but it is too time consuming. Therefor every time you interact with the bulk drive it means and you haven't used the bulk drive in a while you run **autobulk** and provide your password again.

STORE YOUR DATA

You shouldn't store your data in your home folder (*/home/username/*), since this is a small drive. You should save your data to *data1* or *data2*. You all have an own folder with your name there. Showing how full the drives are can be done with **df -h**. A nice way to visualize the data usage on a drive can be done by running **ncdu**. (Another recently added hard drive is *data_nfs* which serves as a stable bridge between sb-ont and basecalling-ont. You can also use this drive for saving stuff, it is only a bit slower.)

PERMISSIONS

Every user is restricted in what they can do. What they can see, remove/change, and execute (resp *rwx* as can be seen with **ls -l**). What permissions apply is determined by the ownership. If you own the file/folder user permissions apply, if you are part of the group, group permissions apply, if not *other* permissions apply. There is a group *brounslab* of which all brounslab people are a member of and this helps in organizing some of these permissions (for example for the package managers).

To elevate your permissions you can prepend a command with **sudo**. Off course you can also do stupid things when being the *super user*, so be carefull when you apply this if you have the right to do this. (**sudo bash** can give a shell with elevated permissions.)

CHANGES TO YOUR ENVIRONMENT

Next to the already mentioned aliases for the bulk drive, other aliases I included to your environment are *interactive* file operations (rm, mv, cp). These will prompt if they need to remove something (rm will always prompt). To not make them prompt you can disable the alias by prepending it with "

Lastly, there are some variables that are assingmed to some useful paths. These are *data1* and *data2* for your personal folder there is *bulk* for your bulk folder. You can use them by prepending them with "$", but cd does not require this.

INSTALLING NEW SOFTWARE (PACKAGE MANAGERS)

There are three package managers on the system. (NB. The name of the package does not always reflect the name of the command.) If you can install with a package manager it is often more easier and keep the dependencies more cleaner. An option for installing with a package manager is not always listed on a software's website, so you can just try.

apt This is the OS package manager. Fast and reliable, but limited amount of niche packages. You can search with **apt search** and list all packages with **dpkg -l**. For installation you need sudo rights, like so: **sudo apt install packagename**.

conda Part of the Anaconda distribution, this is a python based package manager, but it can also install all kinds of other sorts of programs. It also tries to "solve dependencies", but it is often very slow (like really slow). List already installed packages with **conda list**. Searching works with **conda search**, but you often have to include a channel for this. Most likely you need **conda search -c bioconda**.

(Often you are better off with an online search since you don't know the channel.) When installing the channel also needs to be provided. If an installation fails because it cannot find the dependency, try to install this on seperately. The Anaconda distribution also maintains the Perl installation (**which perl**), and therefore you can install perl packages with conda as well.

brew Originally a MacOS package manager, but also works on Linux. Package manager based on git and self-compilation, but can also provide pre-compiled packages. Maintains all dependencies by itself. Self-compilation can be very slow, especially with a lot of dependencies, but often quite fast, especially with pre-compiled packages. List with **brew list** and search **brew search** and install with **brew install**. Channels (or "taps") *brewsci/science* and *brewsci/bio* are already included.

MANUAL INSTALLED

There are a few programs manually installed. They are located in */data1/programs/* and also added to the path. You can just list them with **ls /data1/progams**. You can also add more programs by creating a new folder and install that there. NB. Only the first level is added to the path. So, if you create */data1/programs/newpgrogram/executable*. Only */data1/programs/newprogram* is added to the path, and not *executable*. An exception is made for a bin folder.

To install from source just go to */data1/programs/* (**cd /data1/programs/**) and copy your program to a folder in here. (It really has to be in its own folder.) You can easily download directly from the cluster with **wget http://linktopage.com**. For a git repo, you should use **git clone https://github.com/username/packagename**. This will then make the directory for it by itself.

If you compile from source the only thing you usually have to do it run **make** from within the directory. This only applies if there is *Makefile* in there. If there isn't you sometimes need to run **./configure** before running **make**. Usually just follow the installation instructions on the github page.

Sometimes compilation fails because conda has messed up some environment configurations. You can get a clean environment with the following command:

```
env -i bash --norc --noprofile
```

Usually it will still help to initialize brew again:

```
eval $(/data1/linuxbrew/.linuxbrew/bin/brew shellenv)
```

The binary you end up with in the end has to have the executable permission set. If it hasn't you have to set it yourself with **chmod +x binary**. Also see "permissions" later in this document.

DEPENDENCY MESS

Since everybody (in the brounslab group) can install with the same package managers, and there are two major Python version (Python2 and Python3) with already different installations on the cluster, you can quickly end up in a dependency mess. Conda can deal with most issues, but if you are having trouble take a look at conda environments, or install with a different package manager (like pip). Creating an empty environment can also deal with slow installation of conda packages.

Also, it is good to be aware that currently conda packages come before brew packages in the *$PATH* variable, since Python and Perl can both be installed by brew and conda. On top of this, each programming langeage also has its own package manager.

Python has pip, and pip3. pip3 is always Python3, but pip can be either Python2 or Python3 (check with **pip --version**). Preffered is still conda, since mixing pip and conda too much can lead to problems. If conda does not work, you can try to install with pip.

Perl has multiple ones, but currently using cpanminus (**cpanm**). Initilization happens with **cpanm --local-**

lib=/somefolder/perl5 local::lib. This has already been done and *somefolder* is */home/hwalinga*.) This also requires the line **eval $(perl -I /home/hwalinga/perl5/lib/perl5/ -Mlocal::lib)** in your *.bashrc*, but that is also already done.

WHAT TO USE

I recommend using conda for perl packages, since I think cpanm can have problems.

If you don't know what you are doing I also recommend conda over all other package managers.

If a package has no Python code in it, I think brew is a better alternative if you can find it in there. (In practice very often the case.) Compiling it yourself is also a good alternative if it has no Python code in it.

If a package has only Python code in it, you can also get away by installing with pip, but only if you are sure this package will never be a dependency to another program, but it really is a package on its own. Otherwise dependency mess could happen.

RUNNING LONG JOBS

If you have a long job, you don't want to keep your computer running while it is finishing. Therefore you have to separate such a job from your login session. You can do this with **screen**. Make a new **screen** session with **screen -S sessionname**, start your job and *reattach* the session with CTRL+A+D. You can reattach to your old session with **screen -r sessionname**. With TAB this will autocomplete.

For long jobs it is sometimes a good idea to make the jobs *nice*. This means that they will take less CPU as normally. You can do this by prepending the command it with **nice**, or, when the job is already running open **htop**, find your process and make it **nice** with *f8*. Here you can also kill a process with *F9*. (If the process runs in an interactive session, you can kill it with Ctrl-C.)

GETTING HELP

There are a few ways to get help locally:

tldr With **tldr command** you can get small and quick help with examples.

man With **man command** you get extensive help.

info With **info command** you get even more extensive help (not worth it).

/usr/local/doc
In the folder */usr/local/doc* you can find even more documentation. (Also not worth it.)

There are also various websites you can ask questions, or find answers:

- biostars.org
- reddit.com/r/linuxquestions
- reddit.com/r/bash
- bioinformatics.stackexchange.com
- unix.stackexchange.com
- askubuntu.com
- superuser.com
- serverfault.com
- stackoverflow.com

All websites try to answer a specific niche of question so try the one you think can work best. The people on the websites of the bottom of the list can sometimes be a little mean, but trying can never hurt.

ADDING NEW USERS

There is simple script that adds a new user to the **brounslab** group and also appends the important lines to his/her .bashrc. Script is in */home/hwalinga/bin/newuser* or */data1/hwalinga/bin/newuser*. This can be run with **bash newuser username** (already done for existing users).

TROUBLESHOOTING

When dealing with errors, try to see if you understand the error message before trying some random stuff. This section will probably expand over the months when I am still here.

PYTHON2 VS PYTHON3

When calling a particular python program, be explicit with the Python version (2 vs 3). Use python2 or python3 explicitly. This can also be changed for the first line of the script if such a script uses that (**#!/usr/bin/env python**). If this is not possible you can temporarily create a virtual environment by prepending a self-made folder to the $PATH variable (last resort). Or use conda environments (some thing). Like so:

```
mkdir /home/user/myenv/
ln -s /specific/python /home/user/meyenv
export PATH="/home/user/myenv:$PATH"
```

Indication of wrong Python version are ’print "string"’ vs print("string"), or bytes vs string object.

UNABLE TO FIND PROGAM

The tool has a dependency ("program") that it cannot find (probably because it is not installed). Check this by running **which program** (just to be sure). If you cannot find it, **locate program** might show you its location, so that you can add it to the *$PATH*. (NB. **locate** depends on a database, if you can run **sudo updatedb** beforehand to make sure this is up to date.)

MODULE HAS NO ATTRIBUTE / CANNOT IMPORT MODULE

For Python the former indicates that the module is installed, but does not contain a certain submodule (probably becaue the module is out of date). The latter just indicates the module does not exist. For both problems, try to reinstall the module with conda, if this does not work try to install with pip, or create a conda virtual environment. (NB. Other programs might word this differently, perl will mention something about @INC.)

CANNOT FIND FILE

The tool program requires a certain file has input, but it cannot find it there. You might have mistyped the location. Using <TAB> can make sure you never make typos.

MORE USEFUL STUFF

OPENING FILES

Instead of copying files to your local computer to open them, it is off course more convenient to view them without this hassle. To do this you can view them within you command line, with command line utilties, or graphical with the so called X Server.

xdg-open is a program than can open any file with one of the default applications.

Command line utilities

If you want to take a look at a big file (log or results file), you can make use of pager like **less**. (q to quit)

If you want to quickly edit or create a file on the cluster you can make use of **nano**. Write out means safe.

To get a *rough* idea of the image, you can use **imgcat image.png**. (There are some options that might improve it, like -R or -H.)

X Server

X Server is by default installed if your OS is Linux. On a Mac you will need XQuartz (https://www.xquartz.org/), on a Windows machine MobaXterm promises this functionality out of the box, but need to be installed for other SSH clients, like Putty.

You can now open images/pdf/html from inside the linux cluster.

Opening images can now be done with something like eog. Additionally, you can use a file explorer with image preview (my preferred method). You can start with xterm ranger. (xterm is the terminal emulator ranger will start.)

COPYING BIG FILES

If you want to have a progress bar if you copy something. You can make use of **pycp** instead of **cp**.

If you want to be able to restart an interupted copying progress take a look at **rsync**.

AUTHOR

Hielke Walinga (h.walinga@student.tudelft.nl)