Bash and Git

Jake Lane
Monash University, Australia

StarterKit 2024

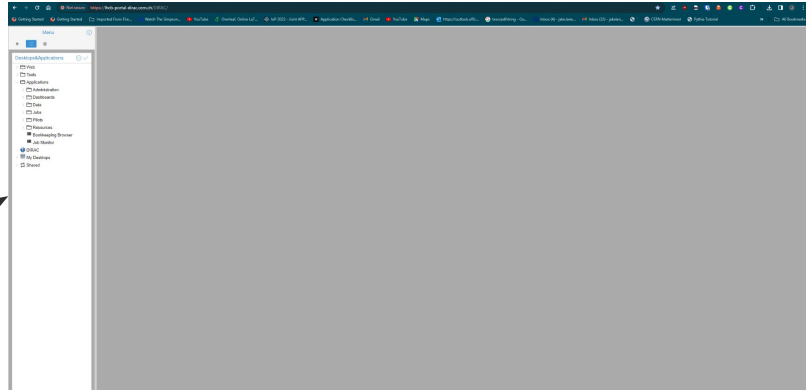
12-16 Feb 2024

Bash

Prerequisites checklist

From the starterkit-lessons :
[Pre-workshop checklist — LHCb Starterkit Lessons documentation]

1. Can you access: https://lhcb-portal-dirac.cern.ch/DIRAC
2. Can you access a terminal and type: `ssh -Y USERNAME@lxplus.cern.ch` with your cern username for `USERNAME`
3. Can you run `lhcb-proxy-init` on `lxplus`?



```
jlane  (e) base  ssh -Y lxplus.cern.ch
Warning: No xauth data; using fake authentication data for X11 forwarding.
* ******************************************************************
* Welcome to lxplus945.cern.ch, Red Hat Enterprise Linux release 9.3 (Plow)
* Archive of news is available in /etc/motd-archive
* Reminder: you have agreed to the CERN
*   computing rules, in particular OC5. CERN implements
*   the measures necessary to ensure compliance.
*   https://cern.ch/ComputingRules
* Puppet environment: production, Roger state: production
* Foreman hostgroup: lxplus/nodes/login
* Availability zone: cern-geneva-a
* LXPLUS Public Login Service - http://lxplusdoc.web.cern.ch/
* Please read LXPLUS Privacy Notice in http://cern.ch/go/7pV7
* 2024-06-27 - lxplus7 CC7 termination https://cern.ch/otg0147045
* ******************************************************************
*****************************************************************************
*                              ---- LbEnv ----                              *
*****************************************************************************
 --- User_release_area is set to /afs/cern.ch/user/j/jolane/cmtuser
 --- CMAKE_PREFIX_PATH is set to:
    /cvmfs/lhcb.cern.ch/lib/lhcb
    /cvmfs/lhcb.cern.ch/lib/lcg/releases
    /cvmfs/lhcb.cern.ch/lib/lcg/app/releases
    /cvmfs/lhcb.cern.ch/lib/lcg/external
    /cvmfs/lhcb.cern.ch/lib/contrib
    /cvmfs/lhcb.cern.ch/lib/var/lib/LbEnv/3067/stable/linux-64/lib/python3.9/site-packages/LbDevTools/data/cmake
-----------------------------------------------------------------------------
[jolane@lxplus945 ~]$
```

```
[jolane@lxplus917 AnalysisProductions]$ lhcb-proxy-init
Generating proxy...
Enter Certificate password: ***************
Added VOMS attribute /lhcb/Role=user
Uploading proxy..
Proxy generated:
subject      : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane/CN=4360923703/CN=3753340571
issuer       : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane/CN=4360923703
identity     : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane
timeleft     : 23:53:58
DIRAC group  : lhcb_user
path         : /tmp/x509up_u118465
username     : jolane
properties   : NormalUser, PrivateLimitedDelegation
VOMS         : True
VOMS fqan    : ['/lhcb/Role=user']

Proxies uploaded:
 DN                                                          | Group | Until (GMT)
 /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane |  | 2024/12/04 08:01
```

lxplus

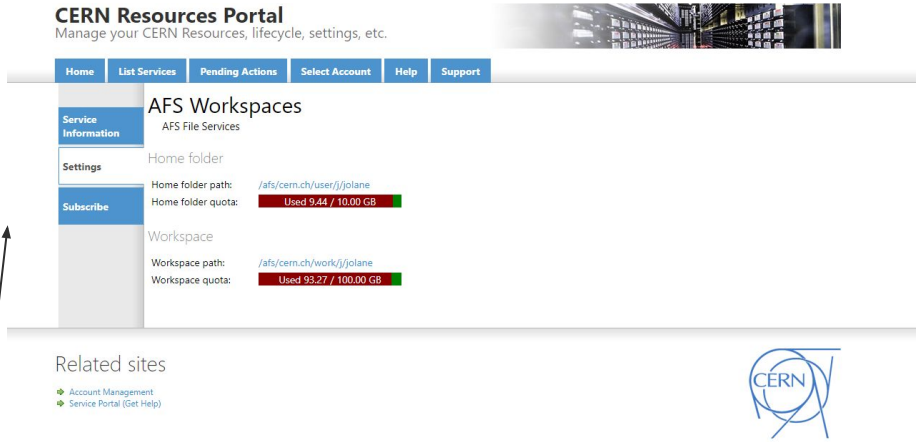
1. **LinuX Public Login User Service https://lxplusdoc.web.cern.ch/**
2. **Linux based machines (based on Red Hat/CentOS/AlmaLinux)**
3. **Different flavours available :**
   - `lxplus7` - Centos7 (used to be default, now legacy)
   - `lxplus8` - AlmaLinux 8
   - `lxplus9` - AlmaLinux 9 - now the default
   - `lxplus-gpu` - GPU node (has AlmaLinux9) equipped with Nvidia Tesla T4

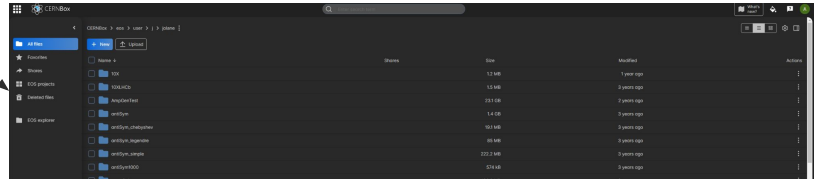
Login via

- `ssh USERNAME@lxplus<7,8,9,-gpu>.cern.ch`
- You actually log in to lxplus9NNN - where NNN is a machine assigned depending on the available resources
- If you have a process running on lxplus800.cern.ch and log out (e.g. with tmux) then you have to log into lxplus800.cern.ch (not lxplus8.cern.ch!)

lxplus Storage

- **Andrew filesystem (AFS) - distributed filesystem for personal files for CERN users, all lxplus nodes look at**
  - /afs/cern.ch/user/u/username (you get up to 10GB here)
  - /afs/cern.ch/work/u/username (LHCb users get 100GB here)
  - Increase your quota https://resources.web.cern.ch/resources/Manage/AFS/
  - Files are backed up (up to 24 hours) in `/afs/cern.ch/ubackup/<initial>/<username>`
- **EOS - (Eos Open Storage) - CERN's filesystem for larger storage**
  - `/eos/user/<initial>/<username>` (you get 1000GB here)
    - Also appears in CERNBOX - like Dropbox for CERN
    - https://cernbox.cern.ch/
  - `/eos/lhcb/user/<inital>/<username>` (LHCb users get 2000GB here)
  - CERNBOX also backs up in
- **CVMFS - (CERN Virtual Machine file system) - contains software used by lxplus**
  - Most programs you run will look at CVMFS
  - Can configure your lxplus session with different versions of common software (e.g. python versions)



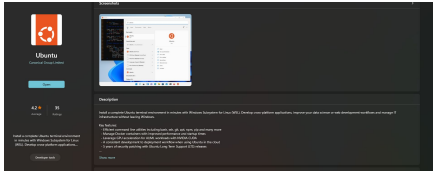


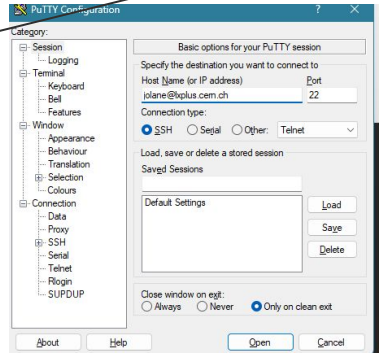
bash

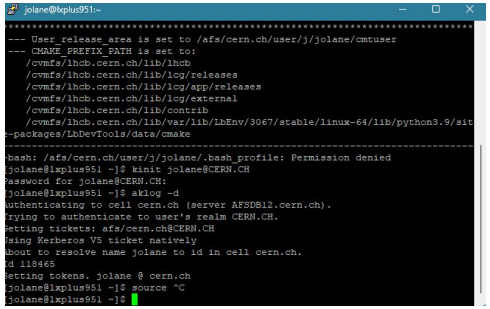
- Bourne Again SHell (`bash`) - GNU version of the Bourne Shell (`sh`), default in UNIX systems (Linux, macOS etc.)
- “Shell program” - used to launch other programs
- On macOS/Linux, just launch any terminal app and you have a bash shell
- There are other options (`zsh`, `csh`, `ksh`,`fish` etc.) but bash is the most popular

bash on Windows

- **Windows doesn't come with `bash` by default**
  - Can install one on Windows 11 - install Terminal, enable WSL, install Ubuntu, launch terminal, set username/password, and you're done!
    - see [Install WSL | Microsoft Learn]() (basically open a command prompt as admin, then type `wsl --install` then restart)
  - OR install WSL manually (Windows 10 - install Ubuntu and enable WSL, launch Ubuntu and set username/password):
    - [Manual installation steps for older versions of WSL | Microsoft Learn]()
  - OR set up a virtual machine (e.g. with VirtualBox) and install Linux there to get a bash terminal
    - https://www.virtualbox.org/
  - OR use PUTTY to SSH into lxplus from Windows directly
    - https://putty.org/







Common bash commands

- **`ls <directory>` (show contents of a directory)**
  - `ls -l -h` shows the contents in a list and in a human-readable format for sizes
  - `ls -a` shows *all* files
  - "." is the current directory, ".." is the directory above
- **`pwd` (print working directory)**
  - `pwd -P` shows the full physical path (e.g. if you set up a symlink, pwd -P bypasses it) of where you are (the working directory)
- **`ln` (link)**
  - symlinks one place in the filesystem with another (effectively a shortcut), e.g. "`ln -s /eos/lhcb/user/u/username/my_analysis/my_big_tuple.root .`" puts "my_big_tuple.root" in your working directory
- **`touch <filename>`**
  - make an empty file called "filename"
- **`cp <target> <destination>`**
  - Copies a file from <target> to <destination>
  - "`cp -pr`" copies a folder and its contents
  - Or do "`rsync -pr -progress`" gives you a progress bar and lets you copy from ssh
- **`rm <target>`**
  - Removes a target file *permanently*
  - "`rmdir`" removes an *empty* directory
  - "`rm -rf`" removes *everything* from the target (be very careful if you use this)

More common bash commands

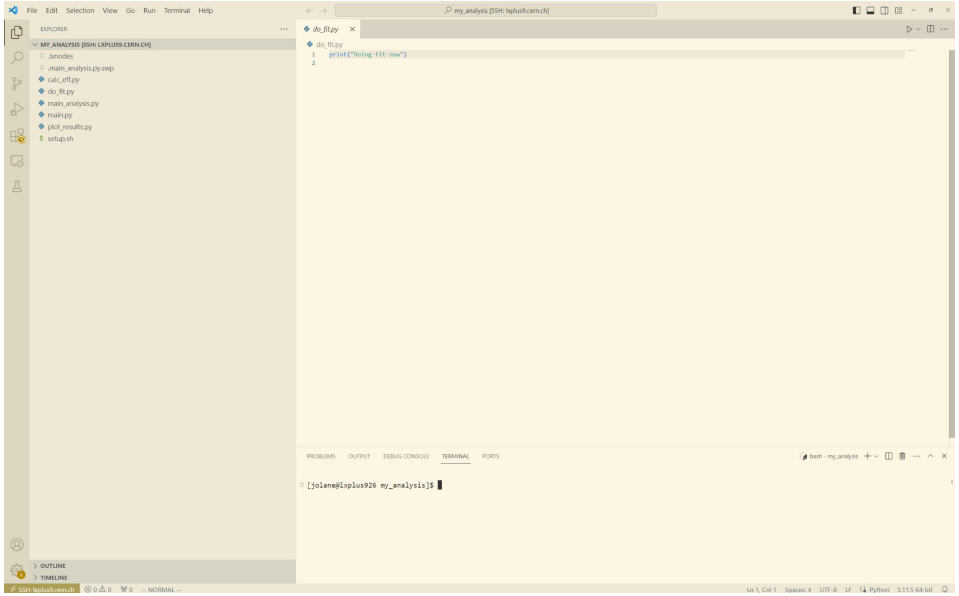
- **echo <argument>**
  - Prints out <argument>, expanding out all variables (e.g. “echo $USER$” should print out your username)
  - Good for testing bash scripts (e.g. echo <my_command> to make sure <my_command> does what you think it will)
- **grep <pattern> <file>**
  - Looks for a pattern in a file - very powerful also combined with pipes
- **find <directory> -n<name>**
  - Finds a file with <name> in <directory> (and subdirectories)
- **sed “s/<find>/<replace>/g” <file>**
  - Powerful find replace tool for any text file - excellent if you need to rename a variable in a big script
  - Do “sed -ie ‘s/<find>/<replace>/g’ <file>” to make a backup of the file before replacing
  - This is also built into “vim”
- **Pipes**
  - The “|” character “pipes” the output from one command to another
  - E.g. “ls . | grep <pattern>” shows all files/folders with <pattern> in them
  - Can be combined with sed
  - Frequently you’ll want to redirect the output to a file - do this with
    - <command> | tee <output>
    - <command> 1> <std_output> 2> <std_err>
- **Loops**
  - Very easy to make loops in bash
  - for i in {0..10}; do <insert code>; done
  - for i in {a,b,c}; do <insert code>; done
  - while [<condition>]; do <insert code>; done
- **Conditions**
  - if [<condition>]; then <commands>; else <commands> ; fi
- **Newlines are specified by “;” and are very important for loops/if statements**
- **Spaces in the [] for conditions matter too!**

More bash tips

- **`. or $PWD`**
  - The current working directory
  - `..` is the directory above the working directory
- **Keyboard shortcuts**
  - If you press the `TAB` key when you are typing a command, bash will try to autocomplete
  - If you press `CTRL+R` you can search for your previous bash commands
  - `CTRL+C` is an interrupt and will stop whatever program you are running,
  - `CTRL + SHIFT + C` and `CTRL + SHIFT + V` are the copy and paste commands
  - `CTRL + Z` suspends a process
- **Monitoring**
  - `ps`, `top` and `htop` all show the current running processes on your shell (and the corresponding process id)
  - `kill <pid>` - kills a process with process id `<pid>`
    - There are different types of "`kill`" - default is `SIGTERM` (terminate the program)
    - `SIGKILL` (or "`kill -9`") kills the process - last resort if you have files opened by that program (try to avoid this command in general)
    - When you log out, programs are `SIGTERM`'ed

Editing files

- **Terminal based editors (good for editing a single file)**
  - nano (easiest)
  - vim
  - Emacs
- **GUI text editors**
  - Emacs has a GUI option as well
  - gvim - GUI version of vim
  - gedit - Linux text editor (basically notepad)
  - Notepad
- **IDE (Integrated Developer Environment)**
  - VS Code
  - Combines shell, file explorer and text editor in one
  - Can also set up remote editing - very useful for unstable connections



Shell scripts

- **Plain text file - execute many bash commands sequentially from a single command**
- **The `#!` ("shebang") tells the prompt (`bash`) to use a specific program to interpret the text (e.g. `#!/bin/python3` executes `python` code)**
  - Tip : do `#!/usr/bin/env <program>` if you want the version of the program that `which program` gets you (mainly for `python` not bash)
- **Make sure that your script is "executable"**
  - `chmod +x my_script.sh`
  - `./my_script.sh`

Shebang - specifies the program

```
#!/usr/bin/env bash
set -eu -o pipefail
shopt -s expand_aliases

#This is a comment
j=20
for i in {0..10};
    do
        echo $i
        (( j+= 1 ))

    done


a=$(( j + 2 ))
echo $j
echo $a
```

Comments have a "#" before them

Assign the output of a command to a variable with $

Loops/if statements need either a ";" or a new line - tabs are optional

Maths operations are done with (( ))

```
jlane  (e) base  ./my_script.sh
0
1
2
3
4
5
6
7
8
9
10
31
33
```

Safety options for scripts

- -u : any undefined variables (e.g. $MYANALYSISDIR) are treated as errors and the script will stop when encountered
- -e if any commands in the script fail, the script immediately fails
- -o pipefail prevents the script from running in pipes if it crashes
- You can also have `set -eux` which will print out every command the script executes (good for debugging)
- Basic maths operations are done with two brackets
- If statements are specified with square brackets

```bash
#!/usr/bin/env bash
set -eux -o pipefail
shopt -s expand_aliases

#This is a comment
j=20
for i in {0..10};
    do
        (( j+= 1 ))


    done


a=$(( j + 2 ))
echo $j
echo $a


if [ -f $HOME/.bashrc ];
    then
        echo Have .bashrc
fi


if (( a==32 ))
    then
        echo $a = 32
fi


if [[ -f $HOME/.bashrc ]]
    then
        echo Still have .bashrc
fi
```

```
jlane  (e) base  ~  ./my_script.sh
+ shopt -s expand_aliases
+ j=20
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ for i in {0..10}
+ ((  j+= 1  ))
+ a=33
+ echo 31
31
+ echo 33
33
+ '[' -f /home/jlane/.bashrc ']'
+ echo Have .bashrc
Have .bashrc
+ ((  a==32  ))
+ [[ -f /home/jlane/.bashrc ]]
+ echo Still have .bashrc
Still have .bashrc
```

Bash environment scripts

- In addition to being mini programs, you can also program your bash script to set up other programs
- For example you might want to set up your grid proxy and python environment (either through LCGViews or conda) and set up some functions
  - So when you log in you just do:
    `cd $HOME/work/my_analysis`
    `source setup.sh`
- You don't need to have the `!#` or do `chmod +x` for these scripts
- You can also "chain" these types of scripts together
- All the loops/if statements work here too
- When you login, `bash` will `source` `$HOME/.bashrc` so if you want to change default behaviour edit that file

```
export ANALYSIS_DIRECTORY=$HOME/work/my_analysis
alias my_main_script="python3 main.py"
kinit jolane@CERN.CH
lhcb-proxy-init
source /cvmfs/sft.cern.ch/lcg/views/setupViews.sh LCG_104 x86_64-el9-gcc12-opt
function run_analysis(){
    python3 do_fit.py
    python3 calc_eff.py
    python3 main_analysis.py
    python3 plot_results.py
}
```

```
[jolane@lxplus925 my_analysis]$ source setup.sh
Password for jolane@CERN.CH:
Generating proxy ...
Enter Certificate password: ***************
Added VOMS attribute /lhcb/Role=user
Uploading proxy..
Proxy generated:
subject      : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane/CN=7842928710/CN=3563520093
issuer       : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane/CN=7842928710
identity     : /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane
timeleft     : 23:53:59
DIRAC group  : lhcb_user
path         : /tmp/x509up_u118465
username     : jolane
properties   : NormalUser, PrivateLimitedDelegation
VOMS         : True
VOMS fqan    : ['/lhcb/Role=user']

Proxies uploaded:
 DN                                                                    | Group | Until (GMT)
 /DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=jolane/CN=833925/CN=Jake Lane |  | 2024/12/04 08:01
```

```
[jolane@lxplus925 my_analysis]$ run_analysis
Doing fit now
Calculating efficiency
Running main analysis
Plotting results
```

.bashrc

- User modification to default bash environment
- Typically used to set up useful aliases, variables
- Can execute programs on launch

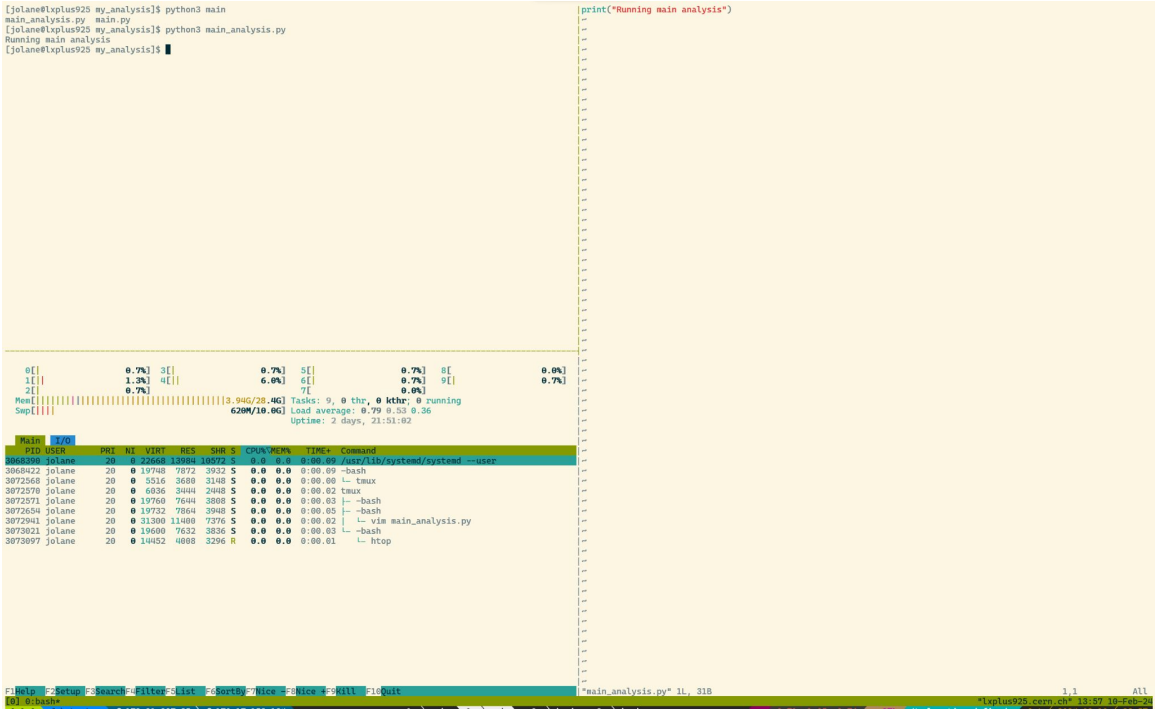
```bash
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions



source $HOME/.bashenv

alias la='ls -a -l -h'
alias ll='ls -l -h'
alias l=ll
function mcd(){
        mkdir $1; cd $1;
}

#


# added by Miniconda3 installer
export CONDAPATH="/afs/cern.ch/user/j/jolane/work/miniconda3/bin"
alias setupConda="export PATH=$PATH:$CONDAPATH"
function snakemake() {
    source "/cvmfs/sft.cern.ch/lcg/views/LCG_93python3/${CMTCONFIG}/setup.sh" && \
    PYTHON3_USER_BASE=$(python3 -m site --user-base) && \
    PYTHON3_USER_SITE=$(python3 -m site --user-site) && \
    export PATH="${PYTHON3_USER_BASE}/bin:${PATH}" && \
    export PYTHONPATH="${PYTHON3_USER_SITE}:${PYTHONPATH}" && \
    "${which snakemake}" "$@"
}

export JULIA_DEPOT_PATH=~/work/julia/.julia:~/work/sw/julia/usr/local/share/julia:~/work/sw/julia/usr/share/julia
export PATH=$HOME/.local/bin:$PATH

ktmux(){
    if [[ -z "$1" ]]; then #if no argument passed
        k5reauth -f -i 36000 -p jolane@CERN.CH -k $HOME/jolane.keytab -- tmux new-session
    else #pass the argument as the tmux session name
        k5reauth -f -i 36000 -p jolane@CERN.CH -k $HOME/jolane.keytab -- tmux new-session -s $1
    fi
}

#function ktmux(){
#    k5reauth -f -i 36000 -p jolane -- tmux
#}

function setupAFS(){
    kinit;
    aklog -d;
    . /afs/cern.ch/user/j/jolane/.bashrc
    lhcb-proxy-init
}

function cleanupGanga(){
    rm /afs/cern.ch/user/j/jolane/work/gangadir/repository/jolane/LocalXML/6.0/sessions/*
}

function setupLCG(){
    source /cvmfs/sft.cern.ch/lcg/views/setupViews.sh LCG_98python3 x86_64-centos7-gcc10-opt
}
```

```bash
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

echo "$(date +[%F_%H:%M]) at $(hostname)" >> .lxnodes

export PATH="$HOME/.cargo/bin:$PATH"
```

This writes the date + time + hostname (specific `lxplus` machine) I logged into to `$HOME/.lxnodes` every time I log into `lxplus`. Useful if I set up a `tmux` session and need to find it.

Tmuxing

- Terminal multiplexers (like `tmux`, `screen`) let you split the session into windows
- And you can log out and still have the system running
- So if you have a program that will take a bit of time to run you *could* use tmux to keep the session running
- Bear in mind that the grid or batch systems like HTCondor are probably what you want to use for big workloads
- Kerberos lets you keep such a session through “keytabs” https://hsf-training.github.io/analysis-essentials/shell-extras/persistent-screen.html
- Lets you keep an eye on CPU/memory usage too (very useful for leaving things running)



Conda and LCGViews

- **Conda** **Installing Miniconda — Anaconda documentation**
  - Easy to install
    - `wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh`
    - `bash Miniconda3-latest-Linux-x86_64.sh`
    - Input install location and say "no" to the prompt at the end
    - `eval $(<install_location>/bin/conda shell.bash)`
    - `conda init`
    - `conda config --set auto_activate_base false`
  - Create python environments in your lxplus directory
    - `conda create -n <env_name> <list_of_packages>`
    - `conda activate <env_name>`
    - `conda install <package>`
- **Generally better to rely on already installed software**
  - See https://cern.service-now.com/service-portal?id=kb_article&n=KB0003076
  - Can get most common software from `LCGViews` https://lcginfo.cern.ch/
  - `source /cvmfs/sft.cern.ch/lcg/views/setupViews.sh <LCG_number> <platform>`
    - Typically the latest number from https://lcginfo.cern.ch/ is fine(e.g. `LCG_104`) and for platform pick
      - `x86_64-el9-gcc12-opt` for `lxplus9` (`gcc` version number will vary)
      - `x86_64-centos7-gcc12-opt` for `lxplus7`
    - Can also get `LCG_104cuda` for `lxplus-gpu` (has gpu supported programs like `tensorflow`)

SSHing

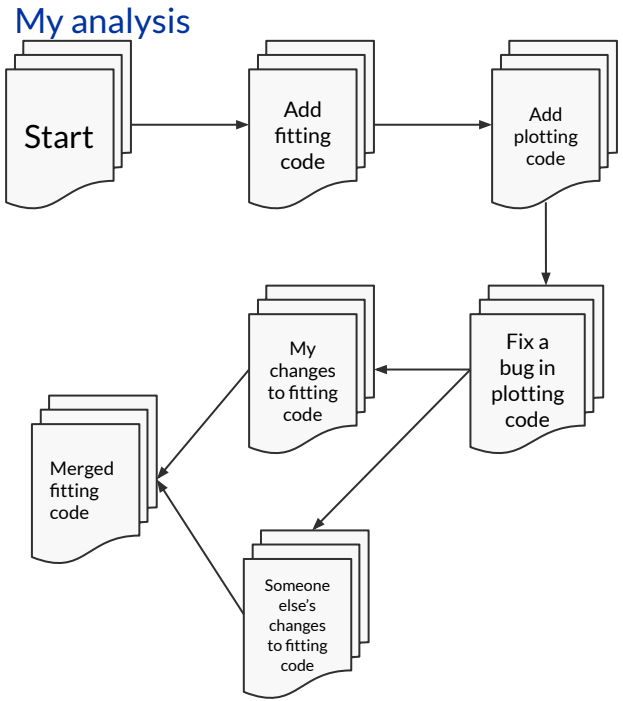
- Passwordless login - do this with `kinit username@CERN.CH`
- Then edit `$HOME/.ssh/config`
- Can also set `user` if your local username is different than the one for lxplus
- Then `ssh cern` should get you into cern also things like `rsync cern:<path_to_file> .` will work
- `GSSAPI` lets you passwordless login and keep permissions on AFS

```
HOST lxplus*
        GSSAPITrustDns yes
        GSSAPIAuthentication yes
        GSSAPIDelegateCredentials yes
        user jolane

HOST cern
        GSSAPITrustDns yes
        GSSAPIAuthentication yes
        GSSAPIDelegateCredentials yes
        hostname lxplus8.cern.ch
        user jolane
```

Git

Git

- **Version control system (VCS) - used to track changes in files for any project (usually computer programs but you can use it for anything: papers, theses etc.)**
- **Can work completely offline or with lots of people over the internet**
- **Most systems (including `lxplus`) have git pre-installed but usually it's best to set it up with your basic info:**
  - `git config --global user.name <your name>`
  - `git config --global user.email <your email>`
- **It will also be useful to set up an ssh keypair for later**
  - `ssh-keygen`
    you can set a password but you don't need to
  - Do this for your actual machine *and* lxplus (you only need to do it once)
- **Sometimes `git` commands don't work if you are using `conda` or `LCGViews` - but default `lxplus` should be fine**

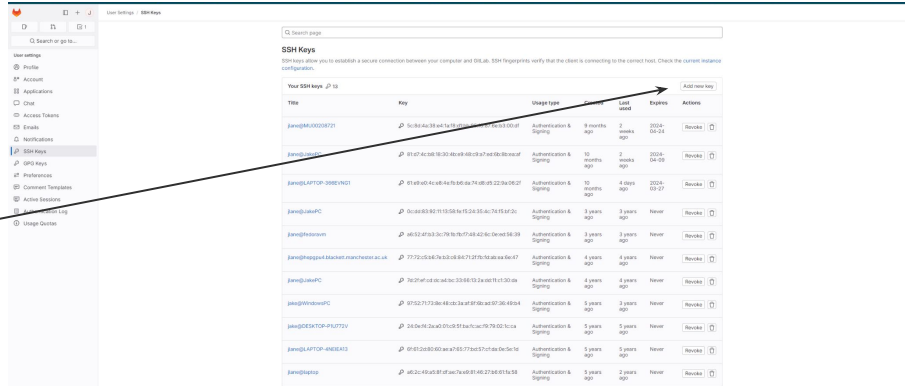


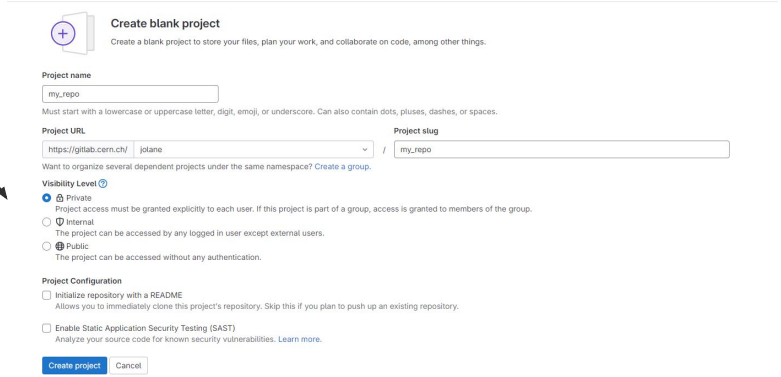
Git basic project

- **Start any git repository (this is what git calls projects) by making a new directory:**
  - `mkdir my_repo`
  - `cd my_repo`
- **Then initialise the repo**
  - `git init`
- **You'll have a folder called ".git" (do "ls -a" to see it) which will contain all the settings for the git repository**
- **You then write files as you would for any other project, e.g.**
  - "my_script.py, my_tuple.root, my_other_script.sh" etc.
- **Then you need to add them to your repository (this "stages" your changes to the repo):**
  - `git add .`
    if you want to add everything in the directory BUT try to avoid this
  - `git add my_script.py my_other_script.sh`
    -usually you don't want to add tuples to git repos (we have /eos/ for that!)
- **Then you "commit" these changes, with a message saying what you did**
  - `git commit -m "Added two scripts"`
    (you can also use a terminal text editor if you do "git commit")

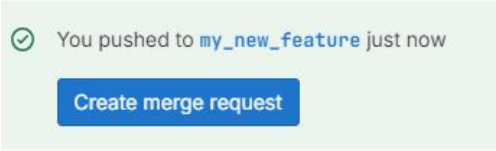
Working with remotes

- Before you go ahead, make sure you have put your *PUBLIC* ssh key in your gitlab/github personal settings
- You copy this from `$HOME/.ssh/id_rsa.pub`
- Do this for every machine (`lxplus` and your own separately) you want to write to `gitlab`
- Go to gitlab.cern.ch and make an empty project there, then clone it to your machine
  - `git clone ssh://git@gitlab.cern.ch:7999/<username>/my_repo.git .`
- Then whenever you've added and committed your code, you just "git push" to change the files on the online repository
- Or from the terminal
  - `git add origin ssh://git@gitlab.cern.ch:7999/<username>/my_repo.git`
  - `git push -u origin`

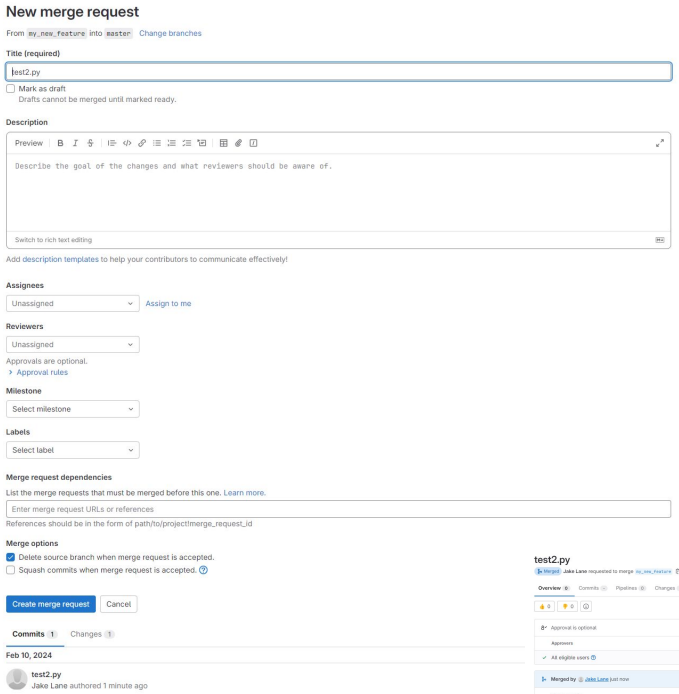




Branches and merge requests



- **On big projects, you can't have everyone just rewriting the entire project - so you usually can't edit the "master" branch of the project**
- **To make changes you typically clone the repository**
  - `git clone ssh://git@gitlab.cern.ch:7990/<username>/my_repo.git`
- **Then you make a new branch**
  - `git branch my_new_feature`
  - `git checkout my_new_feature`
- **Then make your changes locally and "push" using**
  - `git push origin my_new_feature`
- **Then you'll need to make a "merge request" to merge the contents of your new branch with the "master" branch**
  - This is done differently depending on how the project is run/where it's run - often with lots of people asking questions and testing before letting the changes through



`lxplus` specific git

From the StarterKit lessons: Using git to develop LHCb software — LHCb Starterkit Lessons documentation
Main workflow is (e.g. with DaVinci, but you can pick your favourite LHCb software)

- `lb-dev --name DaVinciDev DaVinci/v45r8`
- `cd DaVinciDev`
- `git lb-use DaVinci`
- `git checkout DaVinci/<myPackage>`
- `make`

You can then `git add/commit/push` but this time you'll need to make a merge request with the maintainers of the software you edit
You can submit jobs to the grid with this custom version of DaVinci
You can run the local version with `./run bash --norc` which drops you into a bash session with the custom software installed

- For DaVinci you would do :
  - `./run bash --norc`
  - `gaudirun.py my_tuple_options.py`

Other git properties

- **Rebasing - how “git pull” changes your local version and an updated remote version**
  - Set this with `git config pull.rebase false/true`
  - Setting to true will “rebase” your code - this is sometimes useful to avoid lots of merges/failed pushes but can overwrite your code
  - `git merge` merges two conflicting commits - similar to rebasing
- **`git diff <commit_id> <path>` shows the difference between the file in `<path>` from your version and the one in commit `<commit_id>`**
- **The `.gitignore` file is a special file that you put in the root directory of your repo - you can exclude specific files, files with a particular extension, etc.**
- **CI (continuous integration) - typically used with bigger projects or web facing ones**
  - After a “git push” a script is run on a virtual machine to test any changes made before changing the code
  - Usually not needed in smaller projects
  - Depends on where the project is stored (GitLab v.s. GitHub etc.)
  - Configured by .gitlab-ci.yml in GitLab’s case
  - Both Analysis Productions and Simulation requests do this
- **`git status` can tell you if you have untracked changes**
- **`git restore <commit_id>` restores the branch to the commit with id `<commit_id>`**
- **You can get commit ids from `git history` or using the web GUI**

Commit ids