High Performance Computing for Genomics

Part III: Advanced

Advanced Topics

- **TMP directories**
- pika
- Own module installation

TMP directories

What?

> A tmp or temporary directory or folder is a directory used to hold temporary files. These files are used by some software, and are typically cleaned (removed) after an interval or at completion of the software.

Why?

> To store intermediate files (files used in calculations, but not needed as an end result).

TMP directories

Example: The creation of a bam file typically has a lot of intermediate files:

		Fastq
Mapping	BWA, Bowtie2,...	SAM (reads in order of fastq file)
SAM to BAM conversion	samtools view	BAM (reads in order of fastq file)
Sort according position	samtools sort	BAM (reads in order of position)
Add sample information	Picard AddOrReplaceReadGroups	Regular BAM file (reads in order of position + sample information)

TMP directories

Create temporary files or directories:

```
TMPDIR=$VSC_SCRATCH_NODE;
TMP_DIR=`mktemp -d -t tmp.XXXXXXXX`;
cd $TMP_DIR;
```

Advanced Topics

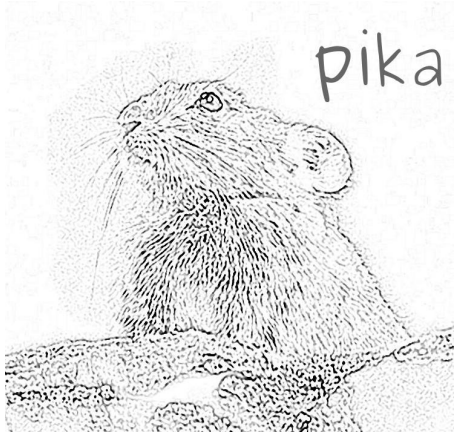
- TMP directories
- **pika**
- Own module installation

pika: *the Pipeline Integration Kit for hpc Analysis*

The pika tool is a help to execute “standard” jobs and pipelines.

pika is a collection of jobs and pipelines, embedded in an environment that helps you to automatically change standard parameters like: project account, mail, project directory, genome directory, …

pika is updated regularly, and freely available on github.
https://github.com/GenomicsCoreLeuven/pika



pika: the tool structure

|-source — This contains all scripts pika uses to run

|-scripts — This contains pbs scripts of jobs, organised in directories according job type.

- |- types
- |- denovo
- |- mapping
- |- variant calling

|-pipelines — This contains files describing the pipelines, including instructions to change the pbs scripts, and bash commands to prepare or review files

pika: Example of a script

```
#!/bin/bash -l
##[RUN] WALLTIME          00:05:00
##[RUN] MEMORY            50gb
##[RUN] NAME              hello_world
##[RUN] ACCOUNT           default_project
##[RUN] NODES             1
##[RUN] CORES_PER_NODE              20
##[VERSION] pika dev [here comes the version of pika where the last changes are made, pika dev for the development fase]
##[HELP] This is the help of the script. Here comes some information for the user on the script.
##
##[HOWTO] #Here follows the howto. To submit the script use:
##[HOWTO] qsub hello_world.pbs
##[HOWTO] #Note that the executable line does not start with #, also in the pipelines and in the tool the ##[HOWTO] will be trimmed.
##
##[HELP] Options:
##[HELP] text: the text to print to the file
##[OPTIONS] text        optional        sed "s:TEXT=\"Hello world\":TEXT=\"value\":g"

#loading the modules
#extra_modules
##here all needed modules will be loaded, handy on 1 place, easy to find the used versions
#setting all parameters (these could be changed)
PROJECT_DIR="";
GENOME_DIR="";
SAMPLE_DIR="$PROJECT_DIR/input";
OUTPUT_DIR="$PROJECT_DIR/output";
SCRATCH_DIR=~;
TEXT="Hello world";

#the actual script
JOBID=""; mkdir -p $SCRATCH_DIR/$JOBID;TMPDIR=$SCRATCH_DIR/$JOBID; TMP_DIR=`mktemp -d -t tmp.XXXXXXXX`;
cd $TMP_DIR;
mkdir -p $OUTPUT_DIR;
echo $TEXT > $OUTPUT_DIR/hello_world.txt;
#here the scratch directory is created, and hello_world is written into a file.
rm -r $TMP_DIR;
```

RUN information headers

Help

Howto

Script Options

Parameters

Code

pika: Example of a pipeline

```
#pipeline template
#=================

##[HELP] The pipeline template
##[HELP] =====================
##[HELP]
##[HELP] This help describes the functions of the pipeline.
##[HELP] It gives a short overview about which steps will be performed on the data.
##[HELP] This help will be shown in the tool when "pipeline help NAME" is given as parameters

##[HOWTO] #This is the howto of the pipeline.
##[HOWTO] #This howto will be added in the pipeline.howto file, created in the jobs directory (with pipeline, as the name of the pipeline)
##[HOWTO] #All commands for the start of the jobs, and short one-liners to edit needed data are inserted in this file

#the job has the exact name of the pbs script.
##[JOB] hello world
#the change follows immediately on the job it applies to. It is change,space,the name of the job,
#tab, the command to add in the pipe (usually a sed, awk, ...)
#In this case, the scratch directory is changed from the node scratch to the test folder in the data directory of the user
##[CHANGE] hello_world        sed "s:SCRATCH_DIR=$VSC_SCRATCH_NODE:SCRATCH_DIR=\"$VSC_DATA/test\":"
#Multiple change commands can follow on the same job.

#Comment in the file can be added, just like this.
#Lines without # are executable lines (just like in bash), example:
cat $VSC_DATA/test/hello_world.txt
```

Help

Howto

Job

Change Job

Bash code

pika: commands

pika [job|pipeline] [list|help|howto|copy] [name]

The copy command for jobs copies the script and modifies it to the copy location (Setting of the project directory, mail, correct headers, ...).

The copy command for pipelines copies all jobs, numbers them and add a pipeline howto, describing the to execute commands.

In both cases extra command line options are possible.

pika

Advantages:

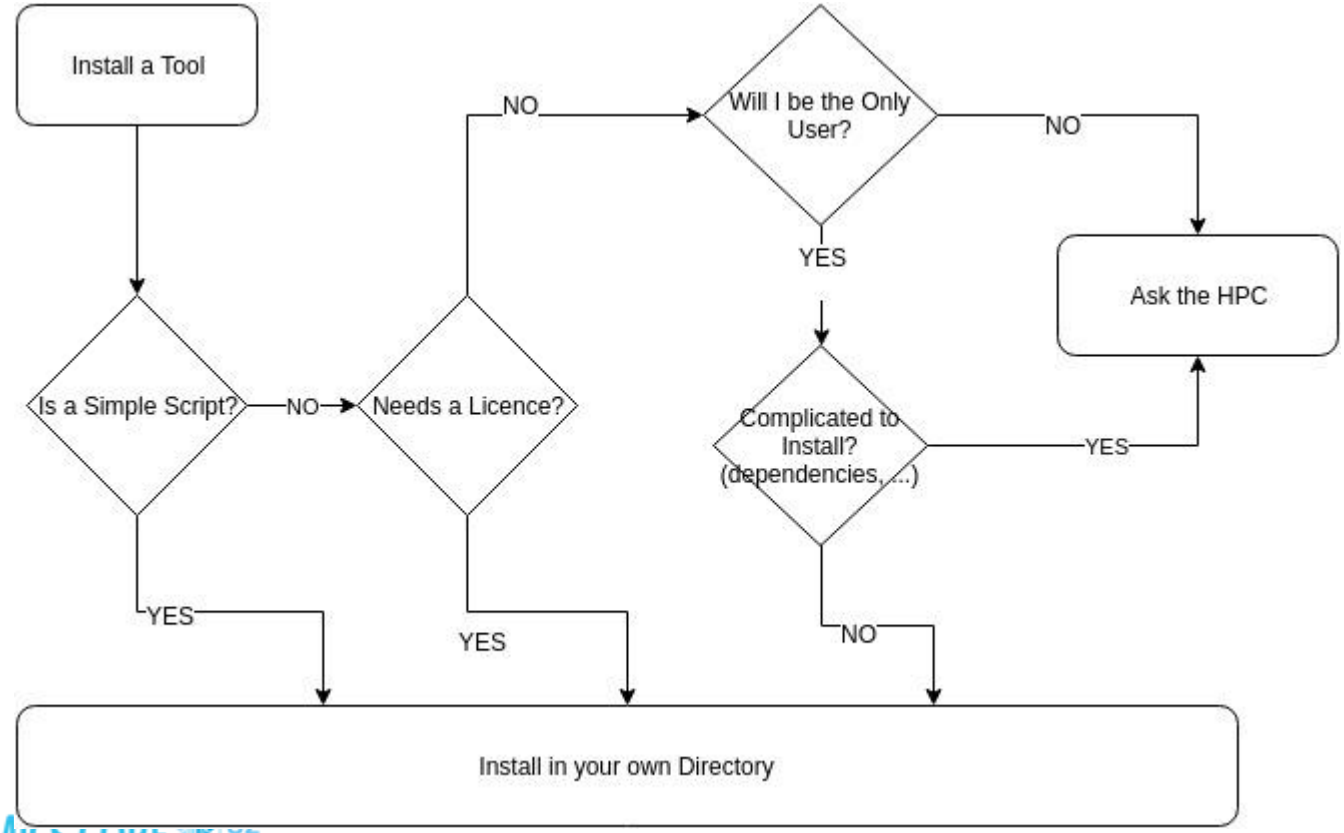
- Available scripts, most already optimized
- You can add own scripts
- You can add own pipelines
- No knowledge of programming language needed (only basic bash)
- Basic new scripts can be made by copying a template, and just pasting your command.

Want to add your script to the pika release? Mail me!

Advanced Topics

- TMP directories
- pika
- **Own module installation**

Installing own modules



Installing own modules

- Create a software directory
- Create a directory with the tool name
- Create a directory with the tool version
  (and additional information like Java, Python, GCC, ... version)
- Install the tool in the directory (or in a lib directory)
- Create a bin directory
- Create links to the executables inside the bin directory (these links will be the names of the tools)
  OR
  Create scripts that launches the tool correctly
- Run the generate_modules script

Installing own modules: Example GBSX

- Goto software folder
- mkdir -p GBSX/v1.3-Java1.8.0_77/lib
- cd GBSX/v1.3-Java1.8.0_77/lib
- wget https://github.com/GenomicsCoreLeuven/GBSX/releases/download/v1.3/GBSX_v1.3.jar
- cd ..
- mkdir bin
- cd bin
- create GBSX file

Installing own modules: Example GBSX: bin/GBSX file

```
#!/bin/bash
JAVA_TMP_DIR="${VSC_SCRATCH}/tmp";
mkdir -p "${JAVA_TMP_DIR}";
java -Djava.io.tmpdir="${JAVA_TMP_DIR}" -Xmx32G -jar
/staging/leuven/stg_00019/software/gbsx/v1.3-Java1.8.0_77/lib/GBSX.jar "${@}";
```

Installing own modules: Example GBSX

- Goto software folder: cd software_folder
- Change permissions:
  chmod -R 666 GBSX
- Generate module:
  generate_modulefile -n GBSX/v1.3-Java1.8.0_77 -s GBSX -v v1.3-Java1.8.0_77 -d “GBSX demultiplexer” -a “source switch_to_2015a” -a “module load Java1.8.0_77”
- Confirm the writing, if the file looks ok
- To use the modules:
- module use software_folder
- module load GBSX/v1.3-Java1.8.0_77

Now you should be ready
for the HPC adventure