Nextflow Hackaton 2017

This workshop will introduce the best practices proposed to tackle the problem of irreproducible NGS data analyses and allow to make large omics workflows portable across HPC clusters and cloud environments.

The practical session will go through the step by step implementation of a pipeline for standard Variant Calling Analyses with RNA-seq data. For this purpose attendees will use Nextflow, a framework for computational pipelines that simplifies the development and deployment of NGS pipelines in a portable manner across different execution environments.

Slides of the workshop can be found at the following links:

- Introduction
- Nextflow

Cheatsheet

A cheatsheet with basic Linux and Cluster commands is available at the end of this document.

1. Data Description

Genome assembly

`genome.fa`

The human genome assembly hg19 (GRCh37) from GenBank, chromosome 22 only.

RNA-seq reads

`ENCSR000COQ[12]_[12].fastq.gz`

The RNA-seq data comes from the human GM12878 cell line from whole cell, cytosol and nucleous extraction (see table below).

The libraries are stranded PE76 Illumina GAIIx RNA-Seq from rRNA-depleted Poly-A+ long RNA (`>200` nucleotides in size).

Only reads mapped to the 22q11 locus of the human genome (`chr22:16000000-18000000`) are used.

ENCODE ID	Cellular fraction	replicate ID	file names
ENCSR000COQ	Whole Cell	1	`ENCSR000COQ1_1.fastq.gz` `ENCSR000COQ1_2.fastq.gz`
		2	`ENCSR000COQ2_1.fastq.gz` `ENCSR000COQ2_2.fastq.gz`

ENCODE ID	Cellular fraction	replicate ID	file names
ENCSR000CPO	Nuclear	1	`ENCSR000CPO1_1.fastq.gz` `ENCSR000CPO1_2.fastq.gz`
		2	`ENCSR000CPO2_1.fastq.gz` `ENCSR000CPO2_2.fastq.gz`
ENCSR000COR	Cytosolic	1	`ENCSR000COR1_1.fastq.gz` `ENCSR000COR1_1.fastq.gz`
		2	`ENCSR000COR2_1.fastq.gz` `ENCSR000COR2_1.fastq.gz`

"Known" variants

`known_variants.vcf.gz`

Known variants come from high confident variant calls for GM12878 from the Illumina Platinum Genomes project. These variant calls were obtained by taking into account pedigree information and the concordance of calls across different methods.

We're using the subset from chromosome 22 only.

Blacklisted regions

`blacklist.bed`

Blacklisted regions are regions of the genomes with anomalous coverage. We use regions for the hg19 assembly, taken from the ENCODE project portal. These regions were identified with DNAse and ChiP-seq samples over ~60 human tissues/cell types, and had a very high ratio of multi-mapping to unique-mapping reads and high variance in mappability.

2. Worflow Description

The aim of the pipeline is to process raw RNA-seq data (in FASTQ format) and obtain the list of small variants, SNVs (SNPs and INDELs) for the downstream analysis. The pipeline is based on the GATK best practices for variant calling with RNAseq data and includes all major steps. In addition the pipeline includes SNVs postprocessing and quantification for allele specific expression.

Samples processing is done **indipendently** for **each replicate**. This includes mapping of the reads, splitting at the CIGAR, reassigning mapping qualities and recalibrating base qualities.

Variant calling is done **simultaniously** on bam files from **all replicates**. This allows to improve coverage of genomic regions and obtain more reliable results.

2.1. Sofware manuals

Documentation for all software used in the workflow can be found at the following links:

- samtools
- picard `CreateSequenceDictionary`
- STAR
- vcftools
- GATK tools
  - `SplitNCigarReads`
  - `BaseRecalibrator`
  - `PrintReads`
  - `HaplotypeCaller`
  - `VariantFiltration`
  - `ASEReadCounter`

2.2. Preparing data

This step prepares input files for the analysis. Genome indexes are created and variants overlapping blacklisted regions are filtered out.

Genome indices with `samtools` and `picard` are produced first. They will be needed for GATK commands such as `Split'N'Trim`:

```
samtools faidx genome.fa
java -jar picard.jar CreateSequenceDictionary R= genome.fa O= genome.dict
```

Genome index for `STAR`, needed for RNA-seq reads mappings, is created next. Index files are written to the folder `genome_dir` :

```
STAR --runMode genomeGenerate \
     --genomeDir genome_dir \
     --genomeFastaFiles genome.fa \
     --runThreadN 4
```

Variants overlapping blacklisted regions are then filtered in order to reduce false positive calls [optional]:

```
vcftools --gzvcf known_variants.vcf.gz -c \
         --exclude-bed blacklist.bed \
         --recode | bgzip -c \
         > known_variants.filtered.recode.vcf.gz
```

2.3. Mapping RNA-seq reads to the reference

To align RNA-seq reads to the genome we're using STAR 2-pass approach. The first alignment creates a table with splice-junctions that is used to guide final alignments. The alignments at both steps are done with default parameters.

Additional fields with the read groups, libraries and sample information are added into the final bam file at the second mapping step. As a result we do not need to run Picard processing step from GATK best practices.

STAR 1-pass:

```
 STAR --genomeDir genome_dir \
      --readFilesIn ENCSR000COQ1_1.fastq.gz ENCSR000COQ1_2.fastq.gz \
      --runThreadN 4 \
      --readFilesCommand zcat \
      --outFilterType BySJout \
      --alignSJoverhangMin 8 \
      --alignSJDBoverhangMin 1 \
      --outFilterMismatchNmax 999
```

Create new genome index using splice-junction table:

```
STAR --runMode genomeGenerate \
     --genomeDir genome_dir \
     --genomeFastaFiles genome.fa \
     --sjdbFileChrStartEnd SJ.out.tab \
     --sjdbOverhang 75 \
     --runThreadN 4
```

STAR 2-pass, final alignments:

```
  STAR --genomeDir genome_dir \
       --readFilesIn ENCSR000COQ1_1.fastq.gz ENCSR000COQ1_2.fastq.gz \
       --runThreadN 4 \
       --readFilesCommand zcat \
       --outFilterType BySJout \
       --alignSJoverhangMin 8 \
       --alignSJDBoverhangMin 1 \
       --outFilterMismatchNmax 999 \
       --outSAMtype BAM SortedByCoordinate \
       --outSAMattrRGline ID:ENCSR000COQ1 LB:library PL:illumina PU:machine
SM:GM12878
```

Index the resulting bam file:

```
samtools index final_alignments.bam
```

2.4. Split'N'Trim and reassign mapping qualities

The RNA-seq reads overlapping exon-intron junctions can produce false positive variants due to inaccurate splicing. To solve this problem the GATK team recommend to hard-clip any sequence that overlap intronic regions and developed a speciall tool for this purpose: `SplitNCigarReads`. The tool identifies Ns in the CIGAR string of the alignment and split reads at this position so that few new

reads are created.

At this step we also reassign mapping qualities to the alignments. This is important because STAR assign the value `255` (high quality) to “unknown” mappings that are meaningless to GATK and to variant calling in general.

This step is done with recommended parameters from the GATK best practices.

```
java -jar GenomeAnalysisTK.jar -T SplitNCigarReads \
                  -R genome.fa -I final_alignments.bam \
                  -o split.bam \
                  -rf ReassignOneMappingQuality \
                  -RMQF 255 -RMQT 60 \
                  -U ALLOW_N_CIGAR_READS \
                  --fix_misencoded_quality_scores
```

2.5. Base Recalibration

The proposed worflow does not include an indel realigment step, which is an optional step in the GATK best practices. We excluded that since it is quite time-intensive and does not really improve variant calling.

We instead include a base recalibration step. This step allows to remove possible systematic errors introduced by the sequencing machine during the assignment of read qualities. To do this, the list of known variants is used as a training set to the machine learning algorithm that models possible errors. Base quality scores are then adjusted based on the obtained results.

```
java -jar GenomeAnalysisTK.jar -T BaseRecalibrator \
                  --default_platform illumina \
                  -cov ReadGroupCovariate \
                  -cov QualityScoreCovariate \
                  -cov CycleCovariate \
                  -knownSites known_variants.filtered.recode.vcf.gz\
                  -cov ContextCovariate \
                  -R genome.fa -I split.bam \
                  --downsampling_type NONE \
                  -nct 4 \
                  -o final.rnaseq.grp

  java -jar GenomeAnalysisTK.jar -T PrintReads \
                  -R genome.fa -I split.bam \
                  -BQSR final.rnaseq.grp \
                  -nct 4 \
                  -o final.bam
```

2.6. Variant Calling and Variant filtering

The variant calling is done on the uniquely aligned reads only in order to reduce the number of false positive variants called:

```
(samtools view -H final.bam; samtools view final.bam| grep -w 'NH:i:1') \
  | samtools view -Sb -  > final.uniq.bam

samtools index final.uniq.bam
```

For variant calling we're using the GATK tool `HaplotypeCaller` with default parameters:

```
ls final.uniq.bam  > bam.list
java -jar GenomeAnalysisTK.jar -T HaplotypeCaller \
                  -R genome.fa -I bam.list \
                  -dontUseSoftClippedBases \
                  -stand_call_conf 20.0 \
                  -o output.gatk.vcf.gz
```

Variant filtering is done as recommended in the GATK best practices:

- keep clusters of at least 3 SNPs that are within a window of 35 bases between them
- estimate strand bias using Fisher's Exact Test with values > 30.0 (Phred-scaled p-value)
- use variant call confidence score `QualByDepth` (QD) with values < 2.0. The QD is the QUAL score normalized by allele depth (AD) for a variant.

```
 java -jar GenomeAnalysisTK.jar -T VariantFiltration \
                  -R genome.fa -V output.gatk.vcf.gz \
                  -window 35 -cluster 3 \
                  -filterName FS -filter "FS > 30.0" \
                  -filterName QD -filter "QD < 2.0" \
                  -o final.vcf
```

2.7. Variant Post-processing

For downstream analysis we're considering only sites that pass all filters and are covered with at least 8 reads:

```
grep -v '#' final.vcf | awk '$7~/PASS/' \
| perl -ne 'chomp($_); ($dp)=$_=~/DP\\=(\\d+)\\;/; if($dp>=8){print $_."\\n"};'
> result.DP8.vcf
```

Filtered RNA-seq variants are compared with those obtained from DNA sequencing (from Illumina platinum genome project). Variants that are common to these two datasets are "known" SNVs. The ones present only in the RNA-seq cohort only are "novel".

Known SNVs will be used for **allele specific expression** analysis.

Novel variants will be used to detect **RNA-editing events**.

We compare two variants files to detect common and different sites:

```
vcftools --vcf result.DP8.vcf --gzdiff known_SNVs.filtered.recode.vcf.gz --diff
-site --out commonSNPs
```

Here we select sites present in both files ("known" SNVs only):

```
awk 'BEGIN{OFS="\t"} $4~/B/{print $1,$2,$3}' commonSNPs.diff.sites_in_files  >
test.bed

vcftools --vcf final.vcf --bed test.bed --recode --keep-INFO-all --stdout >
known_snps.vcf
```

Plot a histogram with allele frequency distribution for "known" SNVs:

```
grep -v '#'  known_snps.vcf | awk -F '\\t' '{print $10}' \
               |awk -F ':' '{print $2}'|perl -ne 'chomp($_); \
               @v=split(/\\,/,$_); if($v[0]!=0 ||$v[1] !=0)\
               {print  $v[1]/($v[1]+$v[0])."\\n"; }' |awk '$1!=1' \
               >AF.4R

gghist.R -i AF.4R -o AF.histogram.pdf
```

Calculate read counts for each "known" SNVs per allele for allele specific expression analysis:

```
java -jar GenomeAnalysisTK.jar -R genome.fa \
               -T ASEReadCounter \
               -o ASE.tsv \
               -I bam.list \
               -sites known_snps.vcf
```

3. Environment Setup

We are going to use Amazon Elastic Compute Cloud (Amazon EC2) virtual machines to work on the practical part of the workshop. Each attendee will run a `t2.large` instance based on a prepared image with the following tools already installed:

- Docker
- Git
- Nano
- Java 8

3.1. Launching the EC2 virtual machine

To connect to the cloud run the following command which launches the virtual machine:

```
eval "$(curl -fsSL https://goo.gl/DdVLrZ)" | bash
```

The above command requires the AWS command line tool which is pre-installed in the workshop's laptops.

```
~~ N G S '1 7  -  W O R K S H O P ~~

Launching EC2 virtual machine
- type  : t2.large
- ami   : ami-61d0ec07
- disk  : 8 GB
- subnet: subnet-05222a43

* Please confirm you want to launch an VM with these settings [y/n]
```

Type `Y Enter` and wait until the instance is ready:

```
* Instance launched >> i-07a9f2ba3795c4f8f <<
* Waiting for ready status ..  [/]
```

3.2. Editing files with GNU nano

While waiting for the virtual machine to be ready we can open a new terminal and have a look at how we can edit files with GNU nano. Check the nano section in the cheatsheet for possible actions.

3.3. Connecting to the virtual machine

Once the previous command is completed and the virtual machine is up and runnning you will get the following message:

```
* The instance is ready -- Login with the following command:

  ssh ngs17@ec2-54-194-213-184.eu-west-1.compute.amazonaws.com
  (password `ngs17`)
```

Follow the on-screen instructions to connect to the machine:

```
ngs17@ec2-54-194-213-184.eu-west-1.compute.amazonaws.com's password:
Last login: Wed Mar 29 12:23:50 2017

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/
7 package(s) needed for security, out of 17 available
Run "sudo yum update" to apply all updates.
[ngs17@ip-172-30-1-84 ~]$
```

3.4. Installing Nextflow

Nextflow can be installed with the following command:

```
curl -fsSL get.nextflow.io | bash
```

```
...
CAPSULE: Downloading dependency com.fasterxml.jackson.core:jackson-
databind:jar:2.6.6
CAPSULE: Downloading dependency ch.qos.logback:logback-classic:jar:1.1.10
CAPSULE: Downloading dependency com.amazonaws:jmespath-java:jar:1.0
CAPSULE: Downloading dependency com.fasterxml.jackson.core:jackson-
annotations:jar:2.6.0

      N E X T F L O W
      Version 0.24.1 build 4245
      last modified 24-03-2017 19:41 UTC
      http://nextflow.io


Nextflow installation completed. Please note:
- the executable file `nextflow` has been created in the folder: /home/ngs17
- you may complete the installation by moving it to a directory in your $PATH
```

Complete the installation copying the `nextflow` launcher file in a folder in the `PATH` variable e.g.:

```
mv nextflow $HOME/bin/
```

To check the installed version use the command below:

```
nextflow info
```

```
  Version: 0.24.1 build 4245
  Modified: 24-03-2017 19:41 UTC
  System: Linux 4.4.51-40.58.amzn1.x86_64
  Runtime: Groovy 2.4.10 on OpenJDK 64-Bit Server VM 1.8.0_121-b13
  Encoding: UTF-8 (UTF-8)
```

3.5. Getting the data

All the files needed for the workshop are stored in a git repository on GitHub. In order to get the data run the following command:

```
git clone https://github.com/CRG-CNAG/ngs2017-nf.git
```

```
Cloning into 'ngs2017-nf'...
remote: Counting objects: 15, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 15 (delta 0), reused 0 (delta 0), pack-reused 12
Unpacking objects: 100% (15/15), done.
Checking connectivity... done.
```

Check the content of the freshly cloned repo:

```
tree ngs2017-nf
```

```
ngs2017-nf
├── bin
│   └── gghist.R
├── data
│   ├── blacklist.bed
│   ├── genome.fa
│   ├── known_variants.vcf.gz
│   └── reads
│       ├── ENCSR000COQ1_1.fastq.gz
│       ├── ENCSR000COQ1_2.fastq.gz
│       ├── ENCSR000COQ2_1.fastq.gz
│       ├── ENCSR000COQ2_2.fastq.gz
│       ├── ENCSR000COR1_1.fastq.gz
│       ├── ENCSR000COR1_2.fastq.gz
│       ├── ENCSR000COR2_1.fastq.gz
│       ├── ENCSR000COR2_2.fastq.gz
│       ├── ENCSR000CPO1_1.fastq.gz
│       ├── ENCSR000CPO1_2.fastq.gz
│       ├── ENCSR000CPO2_1.fastq.gz
│       └── ENCSR000CPO2_2.fastq.gz
├── launch-ec2.sh
└── nextflow.config

3 directories, 18 files
```

3.6. Pulling the Docker image

Nextflow can pull Docker images at runtime, but let's just download it manually to see how Docker works:

```
docker pull cbcrg/callings-
nf@sha256:b65a7d721b9dd2da07d6bdd7f868b04039860f14fa514add975c59e68614c310
```

You should see the progress of the download:

```
sha256:b65a7d721b9dd2da07d6bdd7f868b04039860f14fa514add975c59e68614c310:
Pulling from cbcrg/callings-nf
915665fee719: Downloading [==============================================>     ]
47.08 MB/51.36 MB
f332de2321e6: Downloading [===========>                                       ]
41.96 MB/187.8 MB
1577a6dd9e43: Downloading [================================>                  ]
46.72 MB/73.45 MB
7059d9bb5245: Waiting
71863f70269f: Waiting
ce2a2879246d: Waiting
e38ba5d5f9fb: Waiting
90158da87bb2: Waiting
```

and the following message when the pull is completed:

```
Digest: sha256:b65a7d721b9dd2da07d6bdd7f868b04039860f14fa514add975c59e68614c310
Status: Downloaded newer image for cbcrg/callings-
nf@sha256:b65a7d721b9dd2da07d6bdd7f868b04039860f14fa514add975c59e68614c310
```

4. Pipeline Implementaton

4.1. Data preparation

A first step in any pipeline is to prepare the input data. You will find all the data required to run the pipeline in the folder `data` within the `ngs2017-nf` repository directory.

There are four data inputs that we will use in this tutorial:

1. **Genome File** (`data/genome.fa`)
    - Human chromosome 22 in FASTA file format
2. **Read Files** (`data/reads/`)
    - Sample ENCSR000COQ1: 76bp paired-end reads (`ENCSR000COQ1_1.fq.gz` and `ENCSR000COQ1_2.fq.gz`).
3. **Variants File** (`data/known_variants.vcf.gz`)

- Known variants, gzipped as a Variant Calling File (VCF) format.

4. **Blacklist File** (`data/blacklist.bed`)
    - Genomic locations which are known to produce artifacts and spurious variants in Browser Extensible Data (BED) format.

4.2. Input parameters

We can begin writing the pipeline by creating and editing a text file called `main.nf` from the `ngs2017-nf` repository directory with your favourite text editor. In this example we are using `nano`:

```
cd ngs2017-nf
nano main.nf
```

Edit this file to specify the input files as script parameters. Using this notation allows you to override them by specifying different values when launching the pipeline execution.

```
/*
 * Define the default parameters ①
 */

params.genome     = "data/genome.fa"
params.variants   = "data/known_variants.vcf.gz"
params.blacklist  = "data/blacklist.bed"
params.reads      = "data/reads/ENCSR000COQ1_{1,2}.fastq.gz" ②
params.results    = "results" ③
params.gatk       = "/opt/broad/GenomeAnalysisTK.jar" ④
```

You can copy the above text by using the `Cmd+C` keys, then move in the terminal window, open `nano` and paste the above text by using the `Cmd+V` keys shortcut.

① The `/*`, `*` and `*/` specify comment lines which are ignored by Nextflow.

② The `reads` parameter uses a glob pattern to specify the forward (`ENCSR000COQ1_1.fq.gz`) and reverse (`ENCSR000COQ1_2.fq.gz`) reads are pairs of the same sample.

③ The `results` parameter is used to specify a directory called `results`.

④ The `gatk` parameter specifies the location of the GATK jar file.

Once you have the default parameters in the `main.nf` file, you can save and run the main script for the first time.

With `nano` you can save and close the file with `Ctrl+O`, then `Enter`, followed by `Ctrl+X`.

To run the main script use the following command:

```
nextflow run main.nf
```

You should see the script execute, print Nextflow version and pipeline revision and then exit.

```
N E X T F L O W  ~  version 0.24.1
Launching `main.nf` [nauseous_wright] - revision: 83a0a5415a
```

4.2.1. Problem #1

Great, now we need to transform the parameters into proper file handlers and channel variables. To do that open the `main.nf` file and copy the lines below at the end of the file.

In `nano` you can move to the end of the file using `Ctrl`+`W` and then `Ctrl`+`V`.

This time you must fill the `BLANK` spaces with the correct function and parameter.

```
/*
 *  Parse the input parameters
 */

genome_file     = file(params.genome)
variants_file   = BLANK
blacklist_file  = BLANK
reads_ch        = BLANK
GATK            = params.gatk
```

The first three should specify file objects containing a single `file` as shown in this basic example. For the reads channel, use the fromFilePairs channel factory method. The final one, `GATK` is simply creating a Nextflow variable specifying the path of the GATK application file and does not need any special function.

Once you think you have data organised, you can again run the pipeline. However this time, we can use the the `-resume` flag.

```
nextflow run main.nf -resume
```

See here for more details about using the `resume` option.

4.3. Process 1A

Create a FASTA genome index

Now we have our inputs set up we can move onto the processes. In our first process we will create a genome index using samtools.

You should implement a process having the following structure:

Name

1A_prepare_genome_samtools

Command

create a genome index for the genome fasta with samtools

Input

the genome fasta file

Output

the samtools genome index file

4.3.1. Problem #2

Copy the code below and paste it at the end of `main.nf`.

Your aim is to replace `BLANK` placeholder with the the correct variable name of the genome file that you have defined in previous problem.

```
/*
 * Process 1A: Create a FASTA genome index with samtools
 */

process '1A_prepare_genome_samtools' { ①

  input:
      file genome from BLANK ②

  output:
      file "${genome}.fai" into genome_index_ch ③

  script:
  """
  samtools faidx ${genome} ④
  """
}
```

In plain english, the process could be written as:

① A **process** called 1A_prepare_genome_samtools

② takes as **input** the genome file from `BLANK`

③ and creates as **output** a genome index file which goes into channel `genome_index_ch`

④ **script**: using samtools create the genome index from the genome file

Now when we run the pipeline, we see that the process 1A is submitted:

```
nextflow run main.nf -resume
```

```
N E X T F L O W  ~  version 0.24.1
Launching `main.nf` [adoring_wilson] - revision: 89dbc97b8e
[warm up] executor > local
[17/b0eae4] Submitted process > 1A_prepare_genome_samtools
```

4.4. Process 1B

Create a FASTA genome sequence dictionary with Picard for GATK

Our first process created the genome index for GATK using samtools. For the next process we must do something very similar, this time creating a genome sequence dictionary using Picard.

You should implement a process having the following structure:

Name

1B_prepare_genome_picard

Command

create a genome dictionary for the genome fasta with Picard tools

Input

the genome fasta file

Output

the genome dictionary file

4.4.1. Problem #3

Fill in the `BLANK` words for both the input and output sections.

Copy the code below and paste it at the end of `main.nf`.

Your aim is to insert the correct input name from into the input step (written as `BLANK`) of the process and run the pipeline.

You can choose any channel output name that makes sense to you.

```
/*
 * Process 1B: Create a FASTA genome sequence dictionary with Picard for GATK
 */

process '1B_prepare_genome_picard' {

  input:
      file genome BLANK BLANK

  output:
      file "${genome.baseName}.dict" BLANK BLANK

  script:
  """
  PICARD=`which picard.jar`
  java -jar \$PICARD CreateSequenceDictionary R= $genome O=
${genome.baseName}.dict
  """
}
```

`.baseName` returns the filename without the file suffix. If `"${genome}"` is `human.fa`, then `"${genome.baseName}.dict"` would be `human.dict`.

4.5. Process 1C

Create STAR genome index file

Next we must create a genome index for the STAR mapping software.

You should implement a process having the following structure:

Name

1C_prepare_star_genome_index

Command

create a STAR genome index for the genome fasta

Input

the genome fasta file

Output

a directory containing the STAR genome index

4.5.1. Problem #4

This is a similar exercise as problem 3, except this time both `input` and `output` lines have been left `BLANK` and must be completed.

```
/*
 * Process 1C: Create the genome index file for STAR
 */

process '1C_prepare_star_genome_index' {

  input:
      BLANK_LINE

  output:
      BLANK_LINE

  script:
  """
  mkdir genome_dir

  STAR --runMode genomeGenerate \
       --genomeDir genome_dir \
       --genomeFastaFiles ${genome} \
       --runThreadN ${task.cpus}
  """
}
```

The output of the STAR genomeGenerate command is specified here as `genome_dir`.

4.6. Process 1D

Filtered and recoded set of variants

Next on to something a little more tricky. The next process takes two inputs: the variants file and the blacklist file.

It should output a channel named `prepared_vcf_ch` which emitting a tuple of two files.

In Nextflow, tuples can be defined in the input or output using the `set` qualifier.

You should implement a process having the following structure:

Name

1D_prepare_vcf_file

Command

create a filtered and recoded set of variants

Input

the variants file
the blacklisted regions file

Output

a set containing the filtered/recoded VCF file and the tab index (TBI) file.

4.6.1. Problem #5

You must fill in the two `BLANK_LINES` in the input and the two `BLANK` output files.

```
/*
 * Process 1D: Create a file containing the filtered and recoded set of
variants
 */

process '1D_prepare_vcf_file' {

  input:
      BLANK_LINE
      BLANK_LINE

  output:
      set BLANK, BLANK into prepared_vcf_ch

  script:
  """
  vcftools --gzvcf $variantsFile -c \①
           --exclude-bed ${blacklisted} \②
           --recode | bgzip -c \
           > ${variantsFile.baseName}.filtered.recode.vcf.gz ③

  tabix ${variantsFile.baseName}.filtered.recode.vcf.gz ④
  """
}
```

① The input variable for the variants file

② The input variable for the blacklist file

③ The first of the two output files

④ Generates the second output file named `"${variantsFile.baseName}.filtered.recode.vcf.gz.tbi"`

Try run the pipeline from the project directory with:

```
nextflow run main.nf -resume
```

Congratulations! Part 1 is now complete.

We have all the data prepared and into channels ready for the more serious steps

4.7. Process 2

STAR Mapping

In this process, for each sample, we align the reads to our genome using the STAR index we created previously.

You should implement a process having the following structure:

Name

2_rnaseq_mapping_star

Command

mapping of the RNA-Seq reads using STAR

Input

the genome fasta file
the STAR genome index
a set containing the replicate id and paired read files

Output

a set containg replicate id, aligned bam file & aligned bam file index

4.7.1. Problem #6

Copy the code below and paste it at the end of `main.nf`.

You must fill in the three `BLANK_LINE` lines in the input and the one `BLANK_LINE` line in the output.

```
/*
 * Process 2: Align RNA-Seq reads to the genome with STAR
 */

process '2_rnaseq_mapping_star' {

  input:
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE

  output:
      BLANK_LINE

  script:
  """
  # ngs-nf-dev Align reads to genome
  STAR --genomeDir $genomeDir \
       --readFilesIn $reads \
       --runThreadN ${task.cpus} \
       --readFilesCommand zcat \
       --outFilterType BySJout \
       --alignSJoverhangMin 8 \
       --alignSJDBoverhangMin 1 \
       --outFilterMismatchNmax 999

  # 2nd pass (improve alignmets using table of splice junctions and create a
new index)
  mkdir genomeDir
  STAR --runMode genomeGenerate \
       --genomeDir genomeDir \
       --genomeFastaFiles $genome \
       --sjdbFileChrStartEnd SJ.out.tab \
       --sjdbOverhang 75 \
       --runThreadN ${task.cpus}

  # Final read alignments
  STAR --genomeDir genomeDir \
       --readFilesIn $reads \
       --runThreadN ${task.cpus} \
       --readFilesCommand zcat \
       --outFilterType BySJout \
       --alignSJoverhangMin 8 \
       --alignSJDBoverhangMin 1 \
       --outFilterMismatchNmax 999 \
       --outSAMtype BAM SortedByCoordinate \
       --outSAMattrRGline ID:$replicateId LB:library PL:illumina PU:machine
SM:GM12878

  # Index the BAM file
  samtools index Aligned.sortedByCoord.out.bam
  """
}
```

The final command produces an bam index which is the full filename with an additional `.bai` suffix.

The next step is a filtering step using GATK. For each sample, we split all the reads that contain N characters in their CIGAR string.

4.8. Process 3

GATK Split on N

The process creates k+1 new reads (where k is the number of N cigar elements) that correspond to the segments of the original read beside/between the splicing events represented by the Ns in the original CIGAR.

You should implement a process having the following structure:

Name

3_rnaseq_gatk_splitNcigar

Command

split reads on Ns in CIGAR string using GATK

Input

the genome fasta file
the genome index made with samtools
the genome dictionary made with picard
a set containg replicate id, aligned bam file and aligned bam file index from the STAR mapping

Output

a set containing the sample id, the split bam file and the split bam index file

4.8.1. Problem #7

Copy the code below and paste it at the end of `main.nf`.

You must fill in the four `BLANK_LINE` lines in the input and the one `BLANK_LINE` line in the output.

There is an optional `tag` line added to the start of this process. The `tag` line allows you to assign a name to a specific task (single execution of a process). This is particularly useful when there are many samples/replicates which pass through the same process.

```
process '3_rnaseq_gatk_splitNcigar' {
  tag OPTIONAL_BLANK

  input:
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE

  output:
      BLANK_LINE

  script:
  """
  # SplitNCigarReads and reassign mapping qualities
  java -jar $GATK -T SplitNCigarReads \
                  -R $genome -I $bam \
                  -o split.bam \
                  -rf ReassignOneMappingQuality \
                  -RMQF 255 -RMQT 60 \
                  -U ALLOW_N_CIGAR_READS \
                  --fix_misencoded_quality_scores
  """
}
```

The GATK command above automatically creates a bam index (.bai) of the split.bam output file

---

Next we perform a Base Quality Score Recalibration step using GATK.

4.9. Process 4

GATK Recalibrate

This step uses GATK to detect systematic errors in the base quality scores, select unique alignments and then index the resulting bam file with samtools. You can find details of the specific GATK BaseRecalibrator parameters here.

You should implement a process having the following structure:

Name

4_rnaseq_gatk_recalibrate

Command

recalibrate reads from each replicate using GATK

Input

the genome fasta file

the genome index made with samtools
the genome dictionary made with picard
a set containg replicate id, aligned bam file and aligned bam file index from process 3
a set containing the filtered/recoded VCF file and the tab index (TBI) file from process 1D

Output

a set containing the sample id, the unique bam file and the unique bam index file

4.9.1. Problem #8

Copy the code below and paste it at the end of `main.nf`.

You must fill in the five `BLANK_LINE` lines in the input and the one `BLANK_LINE` line in the output.

```
process '4_rnaseq_gatk_recalibrate' {
  tag "$replicateId"

  input:
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE

  output:
      BLANK into (final_output_ch, bam_for_ASE_ch) ①

  script:
    sampleId = replicateId.replaceAll(/[12]$/,'')
    """
    # Indel Realignment and Base Recalibration
    java -jar $GATK -T BaseRecalibrator \
                  --default_platform illumina \
                  -cov ReadGroupCovariate \
                  -cov QualityScoreCovariate \
                  -cov CycleCovariate \
                  -knownSites ${variants_file} \
                  -cov ContextCovariate \
                  -R ${genome} -I ${bam} \
                  --downsampling_type NONE \
                  -nct ${task.cpus} \
                  -o final.rnaseq.grp

     java -jar $GATK -T PrintReads \
                  -R ${genome} -I ${bam} \
                  -BQSR final.rnaseq.grp \
                  -nct ${task.cpus} \
                  -o final.bam

    # Select only unique alignments, no multimaps
    (samtools view -H final.bam; samtools view final.bam| grep -w 'NH:i:1') \
    |samtools view -Sb -  > ${replicateId}.final.uniq.bam ②

    # Index BAM files
    samtools index ${replicateId}.final.uniq.bam ③
    """
}
```

① The files resulting from this process will be used in two downstream processes. If a process is executed more than once, and the downstream channel is used by more than one process, we must duplicate the channel. We can do this using the `into` operator with parenthesis in the output section. See here for more information on using `into`.

② The unique bam file

③ The index of the unique bam file (bam file name + `.bai`)

Now we are ready to perform the variant calling with GATK.

4.10. Process 5

GATK Variant Calling

This steps call variants with GATK HaplotypeCaller. You can find details of the specific GATK HaplotypeCaller parameters here.

You should implement a process having the following structure:

Name

5_rnaseq_call_variants

Command

variant calling of each sample using GATK

Input

the genome fasta file
the genome index made with samtools
the genome dictionary made with picard
a set containg replicate id, aligned bam file and aligned bam file index from process 4

Output

a set containing the sample id the resulting variant calling file (vcf)

4.10.1. Problem #9

In this problem we will introduce the use of a channel operator in the input section. The groupTuple operator groups together the tuples emitted by a channel which share a common key.

Note that in process 4, we used the sampleID (not replicateID) as the first element of the set in the output. Now we combine the replicates by grouping them on the sample ID. It follows from this that process 4 is run one time per replicate and process 5 is run one time per sample.

Fill in the `BLANKS` as before.

```
process '5_rnaseq_call_variants' {
  tag BLANK

  input:
      BLANK_LINE
      BLANK_LINE
      BLANK_LINE
      BLANK from BLANK.groupTuple()

  output:
      BLANK_LINE

  script:
  """
  echo "${bam.join('\n')}" > bam.list

  # Variant calling
  java -jar $GATK -T HaplotypeCaller \
                  -R $genome -I bam.list \
                  -dontUseSoftClippedBases \
                  -stand_call_conf 20.0 \
                  -o output.gatk.vcf.gz

  # Variant filtering
  java -jar $GATK -T VariantFiltration \
                  -R $genome -V output.gatk.vcf.gz \
                  -window 35 -cluster 3 \
                  -filterName FS -filter "FS > 30.0" \
                  -filterName QD -filter "QD < 2.0" \
                  -o final.vcf
  """
}
```

4.11. Processes 6A and 6B

ASE & RNA Editing

In the final steps we will create processes for Allele-Specific Expression and RNA Editing Analysis.

We must process the VCF result to prepare variants file for allele specific expression (ASE) analysis. We will implement both processes togther.

You should implement two processes having the following structure:

1st process

Name

6A_post_process_vcf

Command

post-process the variant calling file (vcf) of each sample

Input

set containing the sample ID and vcf file
a set containing the filtered/recoded VCF file and the tab index (TBI) file from process 1D

Output

a set containing the sample id, the variant calling file (vcf) and a file containing common SNPs

2nd process

Name

6B_prepare_vcf_for_ase

Command

prepare the VCF for allele specific expression (ASE) and generate a figure in R.

Input

a set containing the sample id, the variant calling file (vcf) and a file containing common SNPs

Output

a set containing the sample ID and known SNPs in the sample for ASE
a figure of the SNPs generated in R as a PDF file

4.11.1. Problem #10

Here we introduce the `publishDir` directive. This allows us to specifiy a location for the outputs of the process. See here for more details.

You must have the output of process 6A become the input of process 6B.

```
process '6A_post_process_vcf' {
  tag BLANK
  publishDir "$params.results/$sampleId" ①

  input:
      BLANK_LINE
      BLANK_LINE

  output:
      BLANK_LINE

  script:
  '''
  grep -v '#' final.vcf | awk '$7~/PASS/' |perl -ne 'chomp($_);
($dp)=$_=~/DP\\=(\\d+)\\;/; if($dp>=8){print $_."\\n"};' > result.DP8.vcf

  vcftools --vcf result.DP8.vcf --gzdiff filtered.recode.vcf.gz  --diff-site
--out commonSNPs
  '''
}


process '6B_prepare_vcf_for_ase' {
  tag BLANK
  publishDir BLANK

  input:
      BLANK_LINE
  output:
      BLANK_LINE
      BLANK_LINE

  script:
  '''
  awk 'BEGIN{OFS="\t"} $4~/B/{print $1,$2,$3}' commonSNPs.diff.sites_in_files
> test.bed

  vcftools --vcf final.vcf --bed test.bed --recode --keep-INFO-all --stdout >
known_snps.vcf

  grep -v '#'  known_snps.vcf | awk -F '\\t' '{print $10}' \
               |awk -F ':' '{print $2}'|perl -ne 'chomp($_); \
               @v=split(/\\,/,$_); if($v[0]!=0 ||$v[1] !=0)\
               {print  $v[1]/($v[1]+$v[0])."\\n"; }' |awk '$1!=1' \
               >AF.4R

  gghist.R -i AF.4R -o AF.histogram.pdf
  '''
}
```

The final step is the GATK ASEReadCounter.

4.11.2. Problem #11

We have seen the basics of using processes in Nextflow. Yet one of the standout features of Nextflow is the operations that can be performed on channels outside of processes. See here for details on the specific operators.

Before we perform the GATK ASEReadCounter process, we must group the data for allele-specific expression. To do this we must combine channels.

The `bam_for_ASE_ch` channel emites tuples having the following structure, holding the final BAM/BAI files:

```
( sample_id, file_bam, file_bai )
```

The `vcf_for_ASE` channel emits tuples having the following structure:

```
( sample_id, output.vcf )
```

In the first operation, the BAMs are grouped together by sample id.

Next, this resulting channel is merged with the VCFs (vcf_for_ASE) having the same sample id.

We must take the merged channel and creates a channel named `grouped_vcf_bam_bai_ch` emitting the following tuples:

```
( sample_id, file_vcf, List[file_bam], List[file_bai] )
```

Your aim is to fill in the `BLANKS` below.

```
bam_for_ASE_ch
  .BLANK                              ①
  .phase(vcf_for_ASE)                 ②
  .map{ left, right ->                ③
    def sampleId = left[0]            ④
    def bam = left[1]                 ⑤
    def bai = left[2]                 ⑥
    def vcf = right [1]               ⑦
    tuple(BLANK, vcf, BLANK, BLANK)   ⑧
  .set { grouped_vcf_bam_bai_ch }     ⑨
```

① an operator that groups sets that contain a common first element.

② the phase operator synchronizes the values emitted by two other channels. See here for more details

③ the map operator can apply any function to every item on a channel. In this case we take our tuple from the phase operation, define the seperate elements and create a new tuple.

④ define repID to be the first element of left.

⑤ define bam to be the second element of left.

⑥ define bai to be the third element of left.

⑦ define vcf to be the first element of right.

⑧ create a new tuple made of four elements

⑨ rename the resulting as `grouped_vcf_bam_bai_ch`

`left` and `right` above are arbitary names. From the phase operator documentation, we see that phase returns pairs of items. So here `left` originates from contents of the `bam_for_ASE_ch` channel and `right` originates from the contents of `vcf_for_ASE` channel.

4.12. Process 6C

Allele-Specific Expression analysis with GATK ASEReadCounter

Now we are ready for the final process.

You should implement a process having the following structure:

Name

6C_ASE_knownSNPs

Command

calculate allele counts at a set of positions with GATK tools

Input

genome fasta file
genome index file from samtools
genome dictionary file
the `grouped_vcf_bam_bai_ch`channel

Output

the allele specific expression file (`ASE.tsv`)

4.12.1. Problem #12

You should construct the process and run the pipeline in its entirety.

```
echo "${bam.join('\n')}" > bam.list

java -jar $GATK -R ${genome} \
                -T ASEReadCounter \
                -o ASE.tsv \
                -I bam.list \
                -sites ${vcf}
```

Congratulations! If you made it this far you now have all the basics to create your own Nextflow workflows.

4.13. Results overview

For each processed sample the pipeline stores results into a folder named after the sample identifier. These folders are created in the directory specified as a parameter in `params.results`.

Result files for this workshop can be found in the folder `results` within the current folder. There you should see a directory called `ENCSR000COQ/` containing the following files:

Variant calls

`final.vcf`

This file contains all somatic variants (SNVs) called from RNAseq data. You will see variants that pass all filters, with the `PASS` keyword in the 7th field of the vcf file (`filter status`), and also those that did not pass one or more filters.

`commonSNPs.diff.sites_in_files`

Tab-separated file with comparison between variants obtained from RNAseq and "known" variants from DNA.

The file is sorted by genomic position and contains 8 fields:

1	`CHROM`	chromosome name;
2	`POS1`	position of the SNV in file #1 (RNAseq data);
3	`POS2`	position of SNV in file #2 (DNA "known" variants);
4	`IN_FILE`	flag whether SNV is present in the file #1 '1', in the file #2 '2', or in both files 'B';
5	`REF1`	reference sequence in the file 1;
6	`REF2`	reference sequence in the file 2;
7	`ALT1`	alternative sequence in the file 1;
8	`ALT2`	alternative sequence in the file 2

`known_snps.vcf`

Variants that are common to RNAseq and "known" variants from DNA.

Allele specific expression quantification

`ASE.tsv`

Tab-separated file with allele counts at common SNVs positions (only SNVs from the file `known_snps.vcf`)

The file is sorted by coordinates and contains 13 fields:

1	`contig`	contig, scaffold or chromosome name of the variant
2	`position`	position of the variant
3	`variant ID`	variant ID in the dbSNP
4	`refAllele`	reference allele sequence
5	`altAllele`	alternate allele sequence
6	`refCount`	number of reads that support the reference allele
7	`altCount`	number of reads that support the alternate allele
8	`totalCount`	total number of reads at the site that support both reference and alternate allele and any other alleles present at the site
9	`lowMAPQDepth`	number of reads that have low mapping quality
10	`lowBaseQDepth`	number of reads that have low base quality
11	`rawDepth`	total number of reads at the site that support both reference and alternate allele and any other alleles present at the site
12	`otherBases`	number of reads that support bases other than reference and alternate bases
13	`improperPairs`	number of reads that have malformed pairs

Allele frequency histogram

`AF.histogram.pdf`

This file contains a histogram plot of allele frequency for SNVs common to RNA-seq and "known" variants from DNA.

4.14. Bonus step

Until now the pipeline has been executed using just a single sample (`ENCSR000COQ1`).

Now we can re-execute the pipeline specifying a large set of samples by using the command shown below:

```
nextflow run main.nf -resume --reads 'data/reads/ENCSR000C*_{1,2}.fastq.gz'
```

It will print an output similar to the one below:

```
N E X T F L O W  ~  version 0.24.1
Launching `main.nf` [backstabbing_nightingale] - revision: 1187e44c7a
[warm up] executor > local
[c6/75e3f4] Submitted process > 1A_prepare_genome_samtools (genome)
[7b/44e5d6] Submitted process > 1C_prepare_star_genome_index (genome)
[da/e19bcf] Submitted process > 1B_prepare_genome_picard (genome)
[95/1ad13d] Submitted process > 1D_prepare_vcf_file (known_variants.vcf)
[72/702900] Submitted process > 2_rnaseq_mapping_star (ENCSR000COR1)
[9a/5ca042] Submitted process > 2_rnaseq_mapping_star (ENCSR000CPO1)
[77/03ef01] Submitted process > 2_rnaseq_mapping_star (ENCSR000COR2)
[04/262db9] Submitted process > 2_rnaseq_mapping_star (ENCSR000COQ2)
[a4/64c69e] Submitted process > 2_rnaseq_mapping_star (ENCSR000CPO2)
[9e/ad3621] Submitted process > 2_rnaseq_mapping_star (ENCSR000COQ1)
[a5/cda1b0] Submitted process > 3_rnaseq_gatk_splitNcigar (ENCSR000COQ2)
[42/0565d7] Submitted process > 3_rnaseq_gatk_splitNcigar (ENCSR000COQ1)
[0c/68ce48] Submitted process > 3_rnaseq_gatk_splitNcigar (ENCSR000COR1)
[6b/3843e1] Submitted process > 3_rnaseq_gatk_splitNcigar (ENCSR000COR2)
[1c/8c474b] Submitted process > 3_rnaseq_gatk_splitNcigar (ENCSR000CPO1)
[98/f17992] Submitted process > 3_rnaseq_gatk_splitNcigar (ENCSR000CPO2)
[c2/8cdfca] Submitted process > 4_rnaseq_gatk_recalibrate (ENCSR000COQ1)
[d1/1a6935] Submitted process > 4_rnaseq_gatk_recalibrate (ENCSR000COR1)
[9f/b4c61d] Submitted process > 4_rnaseq_gatk_recalibrate (ENCSR000COR2)
[aa/b43a43] Submitted process > 4_rnaseq_gatk_recalibrate (ENCSR000COQ2)
[46/2d96f0] Submitted process > 4_rnaseq_gatk_recalibrate (ENCSR000CPO1)
[85/6b9527] Submitted process > 4_rnaseq_gatk_recalibrate (ENCSR000CPO2)
[79/a7fb48] Submitted process > 5_rnaseq_call_variants (ENCSR000CPO)
[a5/29c017] Submitted process > 5_rnaseq_call_variants (ENCSR000COQ)
[22/1fdea2] Submitted process > 5_rnaseq_call_variants (ENCSR000COR)
[7d/e1adfb] Submitted process > 6A_post_process_vcf (ENCSR000CPO)
[0a/4d43fc] Submitted process > 6A_post_process_vcf (ENCSR000COQ)
[18/8d486b] Submitted process > 6A_post_process_vcf (ENCSR000COR)
[60/427153] Submitted process > 6B_prepare_vcf_for_ase (ENCSR000CPO)
[32/64eff0] Submitted process > 6B_prepare_vcf_for_ase (ENCSR000COQ)
[31/32ad40] Submitted process > 6B_prepare_vcf_for_ase (ENCSR000COR)
[6f/a5e211] Submitted process > 6C_ASE_knownSNPs (ENCSR000COR)
[ff/989dc1] Submitted process > 6C_ASE_knownSNPs (ENCSR000CPO)
[25/92875a] Submitted process > 6C_ASE_knownSNPs (ENCSR000COQ)
```

You can notice that this time the pipeline spawns the execution of more tasks because three samples have been provided instead of one.

This shows the ability of Nextflow to implicitly handle multiple parallel task executions depending on the specified pipeline input dataset.

5. NGS 2017 Workshop Cheatsheet

5.1. Basic Linux Commands

5.1.1. Browse the directory structure

`pwd`	tell you where you are
`ls`	list the content of the current directory
`ls <directory>`	list the content of a directory
`cd <directory>`	go to the specified directory
`cd ~ (or cd)`	go to your home directory
`cd ..`	go to the parent directory
`tree <directory>`	list the content of a directory in a tree-like format
`mkdir <directory>`	create the specified directory

5.1.2. View the content of a file

`less, more`	view text with paging
`head`	print first lines of a file
`tail`	print last lines of a file
`cat`	print the content of a file to the screen
`zcat`	print the content of a `gzip` compressed file to the screen

5.1.3. File manipulations

`rm <file>`	remove `file`
`cp <file1> <file2>`	copy `file1` to `file2`
`mv <file1> <file2>`	rename `file1` to `file2`

5.1.4. Some other useful commands

`find <folder>/ -type f`	recursively find all files in a specific folder
`find . -name '<pattern>'`	recursively find anything whose name contains <pattern> in the current folder (Single quotes must be used in order to avoid wildcard expansion by the shell)
`grep <pattern>`	show lines of text containing a given pattern
`grep -v <pattern>`	show lines of text not containing a given pattern
`sort`	sort lines of text files
`wc`	count words, lines and characters
`> (output redirection)`	allow to redirect the output to a file

`| (pipe)`	allow to send the output from one program to another
`cut`	extract selected portion of each line from one or more files
`echo`	input a line of text and display it on standard output

5.1.5. AWK programming

AWK - UNIX shell programming language. A fast and stable tool for processing text files.

`awk '/www/ { print $0 }' <file>`	search for the pattern www in each line of the file
`awk '$3=="www"' <file>`	search for the exact match of www in the third column of the file
`awk 'length($0) > 80' <file>`	print every line in the file that is longer than 80 characters
`awk 'NR % 2 == 0' <file>`	print even-numbered lines of the file

Some built-in variables

`NR`	Number of records
`NF`	Number of fields
`FS`	Field separator character
`OFS`	Output field separator character

See www.grymoire.com/Unix/Awk.html and www.tutorialspoint.com/awk/awk_basic_examples.htm for more information

5.2. Writing and editing files

5.2.1. GNU nano

GNU nano is a text editor for Unix-like operating systems using a command line interface. The following sections describe the commands used for opening, editing and saving files using nano.

File Control

`nano main.nf`	Open or create the file `main.nf`
`kbd:[Ctr+o]` `kbd:[Enter]`	Save changes
`kbd:[Ctrl+x]`	Quit

Navigating through file contents

kbd:[Ctrl+a]	Move to the beginning of the current line
kbd:[Ctrl+e]	Move to the end of the current line
kbd:[Ctrl+v]	Move down one page
kbd:[Ctrl+y]	Move up one page
kbd:[Ctrl+w] kbd:[Ctrl+y]	Go to the beginning of the file
kbd:[Ctrl+w] kbd:[Ctrl+v]	Go to the end of the file

Copy and Paste

kbd:[Ctrl+c] (Mac kbd:[Cmd+c])	Copy the current selection to the clipboard
kbd:[Ctrl+v] (Mac kbd:[Cmd+c])	Paste the contents from the clipboard at the current cursor position
kbd:[Ctrl+k]	Cut from the current cursor position to the end of the current line
kbd:[Ctrl+d]	Delete the character at the current cursor position

Search and Replace

kbd:[Ctrl+w]	Search for a target string