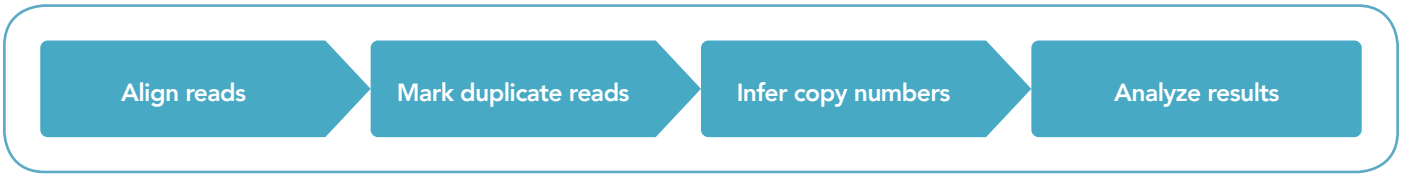
xGen CNV Backbone Panel—Tech Access

Processing sequence data for copy number variations (CNVs)

Introduction

This guideline provides an example workflow for processing next generation sequencing (NGS) data to identify CNVs from FASTQ, or similar raw data. For copy number variation analysis, we recommend a program called CNVkit that allows for CNV calling on single samples (e.g., tumor samples). However, as a best practice, you should build a robust reference with a pool of normal samples to correct for biases in coverage.

Workflow



Note: This example workflow assumes that the sequencing run is demultiplexed using the standard methods recommended for your specific sequencing platform.

Software packages

The following software packages are referenced within our examples:

Software	Version	License	URL
Picard	2.9.0	MIT	https://github.com/broadinstitute/picard
bwa	0.7.15-r1140	GPL3	https://github.com/lh3/bwa
CNVkit	0.9.3	APLv2	http://CNVkit.readthedocs.io/en/stable/
samtools	1.5	CC	http://www.htslib.org/doc/samtools.html

Note: Parameter settings for each tool within a package will affect analysis results. We recommend that you start running individual tools with the "-h" option to view a full list of parameters that you can tune.

Guidelines

Align reads

First, use the Burrows-Wheeler Aligner (BWA)—specifically the BWA-MEM program—to align the reads (FASTQ format) to the reference genome to produce an aligned, sorted-by-position, BAM file. An example invocation follows:

```
bwa mem -t 8 hg19.fa S1_R1.fastq.gz S1_R2.fastq.gz \
| samtools sort -o S1.sorted.bam -
```

Mark duplicate reads

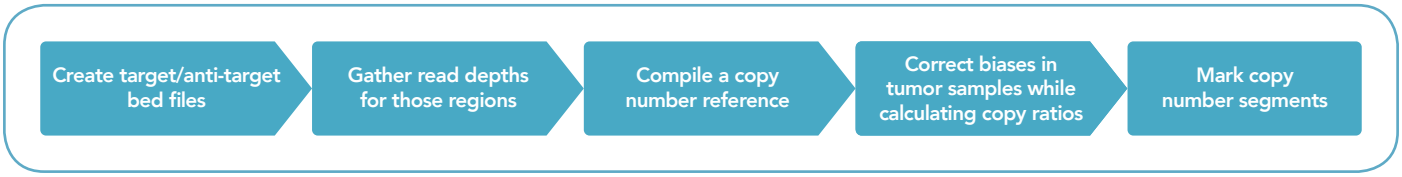
Second, use the MarkDuplicates program (Picard) to identify duplicate reads because CNVkit needs to skip over duplicated reads when calculating read depth. An example invocation follows:

```
java -jar picard.jar MarkDuplicates \
INPUT=S1.sorted.bam \
OUTPUT=S1.Normal.bam \
REMOVE_DUPLICATES=false \
METRICS_FILE=S1.MarkDuplicates.Metrics.txt \
CREATE_INDEX=true
```

Note: The examples in this guide assume that the duplicate marked tumor samples have a "Tumor.bam" suffix, and the duplicate marked normal samples have a "Normal.bam" suffix.

Infer copy numbers using CNVkit

CNVkit has many commands that form the copy number calling pipeline when used together. The general overview of the pipeline is:



CNVkit provides an advantageous way to run the entire pipeline using the *batch* sub-command.

Listed below are two examples of the many ways you could run CNVkit, depending on the number and types of samples you have.

Example 1—Recommended

Build the copy number reference using a pool of normal samples. Infer tumor copy numbers using the *batch* command from CNVkit, as shown:

```
cnvkit.py batch *Tumor.bam --normal *Normal.bam \
--targets baits.bed --fasta hg19.fa \
--access data/access-5kb-mappable.hg19.bed \
--output-reference normal_reference.cnn \
--annotate refFlat.txt \
--output-dir OutputDir/
```

Note: Instead of the target regions, the probes, sometimes referred to as baits, are used in the CNVkit analysis. There is an optional accessibility BED file that describes hard-to-map regions in the genome. CNVkit will take these regions into consideration when making anti-target locations. This file can be found in the data folder of CNVkit and can be invoked using --access. If the targets are missing gene names, you can supply the --annotate parameter with a refFlat file downloaded from the table browser at UCSC.

Example 2

Build a flat copy number reference assuming equal coverage in all probes when no normal samples are available. Infer tumor copy numbers using the *batch* command from CNVkit, as shown:

```
cnvkit.py batch *Tumor.bam --normal
--targets baits.bed --fasta hg19.fa \
--access data/access-5kb-mappable.hg19.bed \
--output-reference flat_reference.cnn \
--annotate refFlat.txt \
--output-dir OutputDir/
```

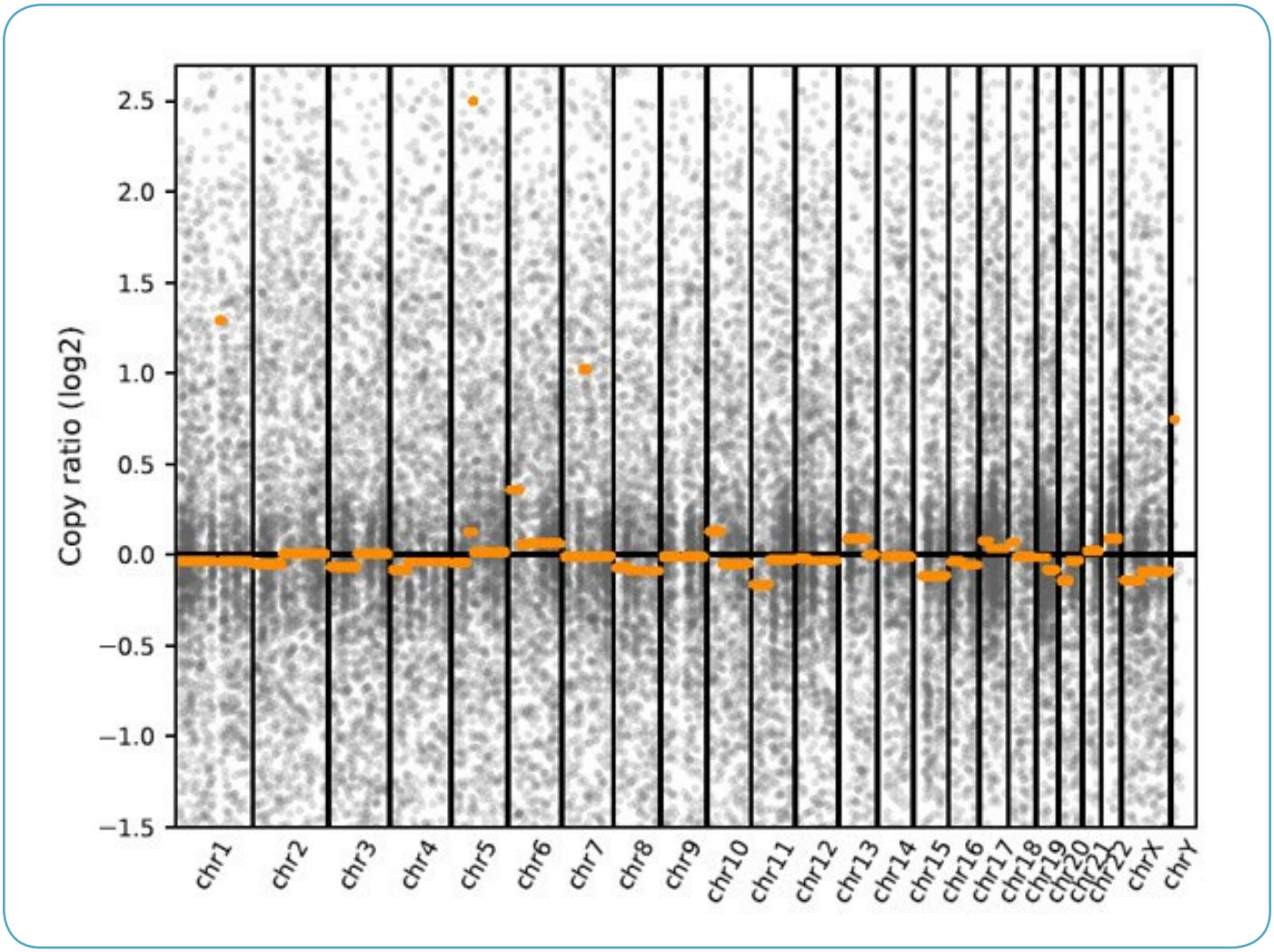
Analyze results

CNVkit has several commands to use for processing the ratio (.cnr) and segment (.cns) resulting from the *batch* command. Several examples are given below.

Example 1

Plot genome-wide copy numbers as a scatter plot. Run the CNVkit *scatter* command using the copy ratio and copy segment files generated for each sample by the previous *batch* command.

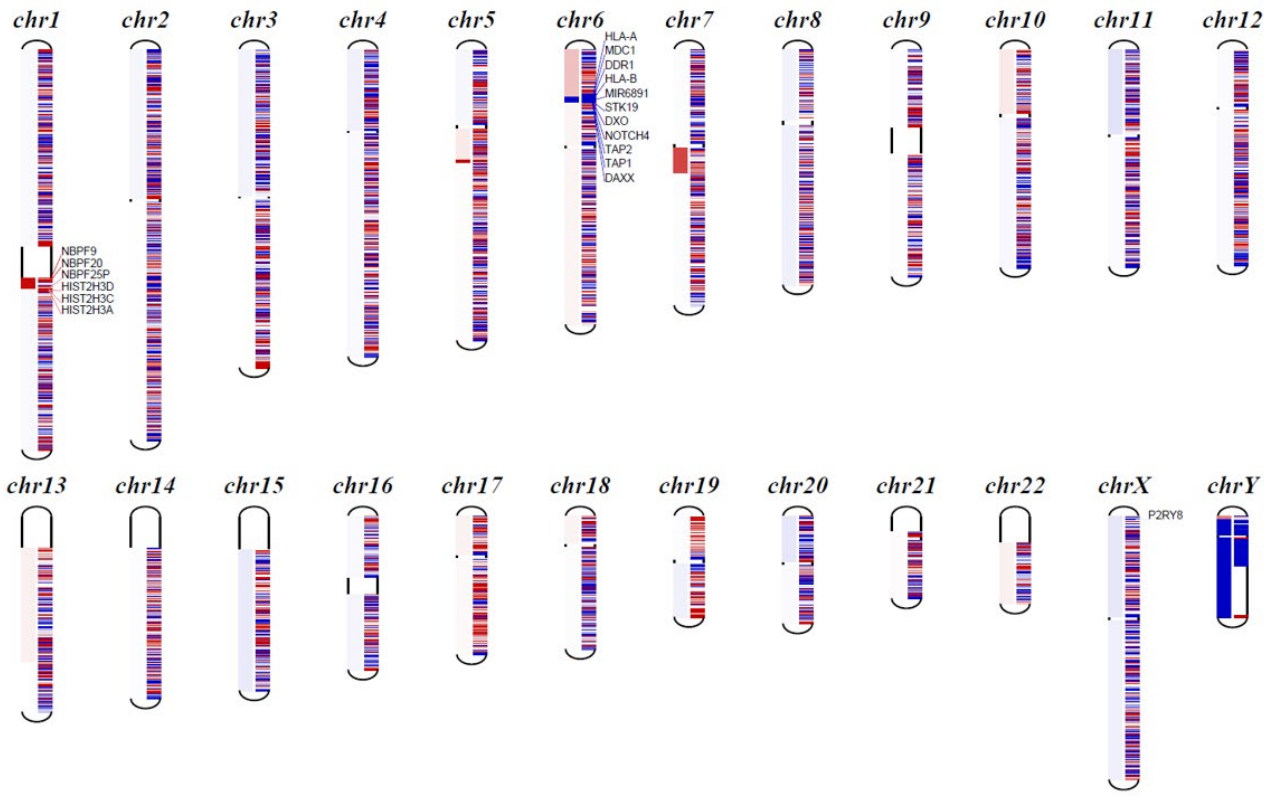
```
cnvkit.py scatter OutputDir/S1.cnr -s OutputDir/S1.cns
```



Example 2

Plot genome-wide copy numbers as an ideogram. Run the CNVkit *diagram* command using the copy ratio and copy segment files generated for each sample by the previous *batch* command.

```
cnvkit.py diagram OutputDir/S1.cnr -s OutputDir/S1.cns
```



Example 3

Identify genes with copy number gained or lost. Run the CNVkit *genemetrics* command using the copy ratio and copy segment files generated for each sample by the previous *batch* command.

```
cnvkit.py genemetrics OutputDir/S1.cnr -s OutputDir/S1.cns \
-t 0.4 -m 5 -o S1.GeneMetrics.txt
```

There are two parameters in the *genemetrics* command that filter which genes are reported:

--threshold (-t) Sets a minimum threshold for the gain/loss $\log_2$ ratio.

--min (-m) Sets a minimum threshold for the number of covered probes needed for a gene to pass the filter.

Inspect the coverage metrics of each sample to determine which samples should be excluded from the analysis because of poor quality, or if another analysis method should be considered (e.g., segmentation algorithms).

Tip: CNVkit provides resource documentation outlining methods, parameters, and in-depth guidelines on their website.

Technical support:

applicationsupport@idtdna.com