UK Biobank WES Protocol / September 2021

The UKB WES protocol describes the single- and aggregate-sample processing, including read alignment, variant calling, joint genotyping and post aggregation reformatting, employed by the Regeneron Genetics Center to generate the UK Biobank WES data set for public release. Following this protocol, researchers can aggregate their own sequencing data with the UK Biobank single sample data, enabling mega-analysis.

1. Aligning whole exome sequencing (WES) reads to a reference genome

The UKB WES data are reference-aligned with the OQFE protocol, which employs BWA MEM to map all reads to the GRCh38 reference in an alt-aware manner, marks read duplicates, and adds additional per-read tags. The OQFE protocol retains all reads and original quality scores such that the original FASTQ is completely recoverable from the resulting CRAM file. All constituent steps of the OQFE protocol are executed with open-source software and described in detail in the OQFE manuscript linked above. Given the impact even small changes to the protocol can introduce into large analyses, the OQFE protocol is available as a Docker file and as an app on DNAnexus that executes the OQFE Docker file[1].

The OQFE Docker file takes either FASTQ or CRAM files as inputs and outputs an OQFE CRAM, ensuring that all steps are executed exactly as specified in the OQFE protocol. We strongly recommend that users seeking to harmonize their data with UKB WES data execute the OQFE protocol via these implementations.

2. Variant calling using DeepVariant

This section of the UKB WES protocol describes the variant calling using DeepVariant (https://github.com/google/deepvariant). To call variants in WES data, either the default DeepVariant WES model or a custom model can be used. The custom model that is trained on WES data generated by Regeneron Genetics Center and used for the generation of UK Biobank data is available as supplementary materials in Krasheninina et al., 2020 (https://www.biorxiv.org/content/10.1101/2020.12.15.356360v1).

2.1 Prerequisites:

a) DeepVariant v0.10.0 docker file
b) OQFE CRAM generated by following section 1 of this protocol (e.g. aligned.cram)
c) GRCh38 reference sequences (e.g. references.fa, detailed in OQFE protocol)
d) Calling regions in BED format (e.g. calling_regions.bed)
e) A custom model file, if applicable

```
docker run deepvariant –model_type=WES –ref=references.fa –reads=aligned.cram –regions calling_regions.bed
–output_gvcf=sample.gvcf –num_shards=<num_threads> --customized_model <model_file>
```

[1] https://hub.docker.com/r/dnanexus/oqfe and https://blog.dnanexus.com/2021-01-21-new-pipeline-public-sequencing-datasets-oqfe/

Command section	Annotation
docker run	Docker run
deepvariant	Deepvariant docker file
–model_type=WES	Whole Exome Sequencing model
–ref=references.fa	Reference sequences
–reads=aligned.cram	Alignment file
–regions calling_regions.bed	Calling regions in BED format
–output_gvcf=sample.gvcf	Output gVCF file
–num_shards=<num_threads>	Number of threads
--customized_model <model_file>	Specify the customized model file, if applicable

3. Aggregation and genotype harmonization using GLnexus to generate a multi-sample project-level VCF (pVCF)

This section of the UKB WES protocol describes the aggregation of variant genotypes and variant allele harmonization across sample-level gVCFs into multi-sample project-level VCF file (pVCF) organized in chromosomes or genomic segments using GLnexus, containing a row for every set of overlapping variants and each sample's genotype for every variant allele. This pVCF is "squared-off", in that for samples that do not contain an alternative allele genotype for a given variant, genotypes are derived from the gVCF reference blocks, reporting the read depth and most likely genotype (i.e. 0/0 or missing) for that sample at that variant position.

The GLnexus aggregation process requires only the gVCFs and the desired aggregation regions in the BED format (BED format specifications, https://m.ensembl.org/info/website/upload/bed.html) as inputs. The BED file for UKB WES data is the exome capture region buffered by 100 bp on each side of each target, with overlapping buffered regions merged. Users who are applying this protocol to a set of gVCFs derived from sequencing across multiple capture designs will need to generate a unified BED file for the regions of interest. The protocol here describes the BED generation process for aggregating variants across multiple capture designs: the intersection of all capture designs and the union of all capture designs.

3.1 Preparation of calling regions in BED format

3.1.1 Scenario 1: Adding 100 bp buffer on each side of the custom target regions in BED format

3.1.1.1 Prerequisites:

a) BEDtools (https://bedtools.readthedocs.io/en/latest/)
b) Custom target regions in BED format (e.g. targets.bed)
c) FASTA index file (.fai) of the reference sequences (e.g. references.fa.fai)

3.1.1.2 Steps:

3.1.1.2.1 Generate 100 bp buffer regions of each side of the target regions

```
bedtools flank -i target.bed -g references.fa.fai -b 100 > buffer.bed
```

Command section	Annotation
bedtools flank	To create flanking intervals for each BED feature
-i target.bed	The input BED file
-g references.fa.fai	The genome file defining chromosome bounds
-b 100	The number of base pairs in each direction to add to the input BED file
> buffer.bed	Redirect the output to the buffer.bed

3.1.1.2.2 Combine the target and buffer regions and sort based on chromosome and start coordinates

```
cat target.bed buffer.bed | sort -k1,1 -k2,2n > target_buffer.bed
```

Command section	Annotation
cat target.bed buffer.bed	Concatenate the target and buffer BED files to the standard output
| sort -k1,1 -k2,2n	Pipe to a sort command to sort the BED by chromosome first and then by the start coordinates in the numeric order
> target_buffer.bed	Redirect the output to a target_buffer.bed file

3.1.1.2.3 Merge overlapping regions

```
bedtools merge -i target_buffer.bed > calling_regions.bed
```

Command section	Annotation
bedtools merge	Merge overlapping BED features into a single interval
-i target_buffer.bed	The input BED file
> calling_regions.bed	Redirect the output to a calling_regions.bed file

3.1.2 Scenario 2: Merging to create the union of two different calling regions in BED format

3.1.2.1 Prerequisites:

a) BEDtools (https://bedtools.readthedocs.io/en/latest/)
b) Two different calling regions in BED format (e.g. calling_regions_1.bed, calling_regions_2.bed) Note: both BED files need to have coordinates of the same genome build and the same chromosome naming convention as the reference sequences.

3.1.2.2 Steps:

3.1.2.2.1 Combine the two calling regions and sort based on chromosome and start coordinates

```
cat calling_regions_1.bed calling_regions_2.bed | sort -k1,1 -k2,2n > combined.bed
```

Command section	Annotation
cat calling_regions_1.bed calling_regions_2.bed	Concatenate the calling_regions_1 and calling_regions_2 BED files to the standard output
| sort -k1,1 -k2,2n	Pipe to a sort command to sort the BED by chromosome first and then by the start coordinates in the numeric order
> combined.bed	Redirect the output to a combined.bed file

3.1.2.2.2 Merge overlapping regions

```
bedtools merge -i combined.bed > combined_calling_regions.bed
```

Command section	Annotation
bedtools merge	Merge overlapping BED features into a single interval
-i combined.bed	The input BED file
> combined_calling_regions.bed	Redirect the output to a combined_calling_regions.bed file

3.1.3 Scenario 3: Create the intersect of two different calling regions in BED format

3.1.3.1 Prerequisites:

c) BEDtools (https://bedtools.readthedocs.io/en/latest/)
d) Two different calling regions in BED format (e.g. calling_regions_1.bed, calling_regions_2.bed) Note: both BED files need to have coordinates of the same genome build and the same chromosome naming convention as the reference sequences.

3.1.3.2 Steps:

3.1.3.2.1 Get the intersect of the two calling regions with BEDtools

```
bedtools intersect -a calling_regions_1.bed -b calling_regions_2.bed > intersect.bed
```

Command section	Annotation
bedtools intersect	Bedtools intersect
-a calling_regions_1.bed	First calling regions in BED format
-b calling_regions_2.bed	Section calling regions in BED format
> intersect.bed	Redirect the output to a intersect.bed file

3.2 Running GLnexus workflow in DNAnexus platform to generate pVCF

3.2.1 Prerequisites:

a) Access to DNAnexus platform and the 'GLnexus workflow' in DNAnexus[2]
b) dx-toolkit (https://documentation.dnanexus.com/downloads) in local environment
c) Access to a DNAnexus folder containing the single sample genomic VCF (gVCF) for all samples to be included in pVCF.
d) A BED format file containing the genomic regions to be aggregated and reported in the pVCF

3.2.2 Steps:

3.2.2.1 Generation of a manifest file containing the DNAnexus file ids of the gVCF files.

```
dx find data --project=<dx_project_name> --folder=<folder_name> --name="*.gvcf.gz" --brief | cut -d\: -f2 > <manifest_file>
```

Command section	Annotation
dx find data	Find data objects subject to the given search parameters
--project=<dx_project_name>	The DNAnexus project name where the data is in
--folder=<folder_name>	The folder where the data is in the DNAnexus project
--name="*.gvcf.gz"	The files to look for using a wildcard "*"
--brief	Print a DNAnexus ID per line for all matching gVCF files
| cut -d\: -f2	Pipe to the cut command to extract the DNAnexus file IDs only
> <manifest_file>	Redirect the output to a file

3.2.2.2 Upload the <manifest_file> to the DNAnexus platform

```
dx upload --path <dx_project>:/<dx_folder>/ <manifest_file> --brief
```

Command section	Annotation
dx upload	Upload local file to DNAnexus
--path <dx_project>:/<dx_folder>/	The folder <dx_folder> in the DNAnexus project <dx_project> that the manifest file is uploaded to
<manifest_file>	The local manifest file that needs to be uploaded
--brief	Return a DNAnexus file ID for the uploaded manifest file

[2] https://github.com/dnanexus-rnd/GLnexus. The GLnexus workflow will shortly be available in the Research Analysis Platform – please refer to the online community forum for more details.

3.2.2.3 Launch GLnexus workflow (version 1.3.1) to generate pVCF files

```
dx run GLnexus_joint_calling_workflow --priority high -y --brief -i common.gvcf_manifest=<manifest_file_location> -i common.targets_bed=<calling_regions.bed> -i unify.shards_bed=<shards_bed> -i etl.shards=20 -i common.config=DeepVariantWES -i common.output_name=<base_name> --folder=<output folder in DNAnexus>
```

Command section	Annotation
dx run	Run a workflow in DNAnexus
GLnexus_joint_calling_workflow	The name of the workflow
--priority high	Set the priority of the workflow run to 'high'
-y	Suppress prompt for confirmation
--brief	Return a DNAnexus analysis ID for the launched workflow
-i common.gvcf_manifest=<manifest_file_location>	Specify the location of the manifest file in DNAnexus. This can be an absolution DNAnexus path like <dx_project>:/<dx_folder>/<manifest_file> or a DNAnexus file ID return in step 2
-i common.targets_bed=<calling_regions.bed>	The calling regions in BED format
-i unify.shards_bed=<shards_bed>	The desired shards in BED format
-i etl.shards=20	The number of shards in the ETL process
-i common.config=DeepVariantWES	The built-in configurations [a]
-i common.output_name=<base_name>	The base name of the output pVCF files
--folder=<output folder in DNAnexus>	The output folder in DNAnexus

Note: [a] the available build-in configurations include DeepVariantWES, DeepVariantWGS, wecall, and gatk

4. Conversion of pVCF to PLINK and BGEN files

This section of the UKB WES protocol describes how UKB WES PLINK and BGEN format files are derived from the pVCF, including the decomposition of multi-allelic variants into biallelic variants and variant normalization prior to format conversion to PLINK and BGEN. BGEN is recommended for running Regenie (https://doi.org/10.1038/s41588-021-00870-7).

4.1 Prerequisites:

a) bcftools (http://www.htslib.org/download/)
b) PLINK 1.9
c) PLINK 2.0 (for BGEN conversion)
d) QCTOOL (https://www.well.ox.ac.uk/~gav/qctool_v2/)
e) Reference sequences in FASTA format (e.g. references.fa)

4.2 Steps:

4.2.1 Split multiallelic variants in pVCF file and variant normalization.

```
bcftools norm -f references.fa -m -any -Oz -o pvcf.norm.vcf.gz pvcf.vcf.gz
```

Command section	Annotation
bcftools norm	Split multiallelic variants and normalization
-f references.fa	Specify the reference sequences (required for left-alignment and normalization)
-m -any	Split any multiallelic variants
-Oz	The output type is 'compressed VCF'
-o pvcf.norm.vcf.gz	Write output to a file named 'pvcf.norm.vcf.gz'
pvcf.vcf.gz	The input pVCF file

4.2.2 Convert normalized biallelic pVCF to PLINK files.

```
plink --vcf pvcf.norm.vcf.gz --keep-allele-order --vcf-idspace-to _ --double-id --allow-extra-chr 0 --make-bed --vcf-half-call m --out pvcf.norm
```

Command section	Annotation
plink	PLINK 1.9
--vcf pvcf.norm.vcf.gz	Specify the input pVCF file
--keep-allele-order	Keep the allele order
--vcf-idspace-to _	Change spaces in the variant IDs to underscore (_)
--double-id	Set both family ID and within-family ID to the same sample ID
--allow-extra-chr 0	Specify the input pVCF file
--make-bed	Set unrecognized chromosome codes to 0
--vcf-half-call m	Convert half calls (./1) in the pVCF to missing in PLINK
--out pvcf.norm	Specify the prefix of the output PLINK

4.2.3 Convert PLINK files to BGEN files and prepare BGEN files

4.2.3.1 Convert PLINK to zlib-compressed BGEN file

```
plink2 --bfile pvcf.norm --export bgen-1.2 bits=8 ref-first --out pvcf.norm_zlib
```

Command section	Annotation
plink2	PLINK2
--bfile pvcf.norm	The input genotype file
--export bgen-1.2 bits=8 ref-first	The output format bgen with 8-bits probability precision and correct allele order
--out pvcf.norm_zlib	The output genotype file

4.2.3.2 Convert zlib-compressed BGEN to zstd-compressed BGEN file using QCTOOL

```
qctool -g pvcf.norm_zlib.bgen -s pvcf.norm_zlib.sample -og pvcf.norm.bgen -os pvcf.norm.sample -ofiletype bgen
-bgen-bits 8 -bgen-compression zstd -bgen-omit-sample-identifier-block
```

Command section	Annotation
qctool	QCTOOL v2
-g pvcf.norm_zlib.bgen	The input genotype file
-s pvcf.norm_zlib.sample	The input sample file
-og pvcf.norm.bgen	The output genotype file
-os pvcf.norm.sample	The output sample file
-ofiletype bgen	The filetype of the output genotype file specified by -og
-bits 8	Store each probability in 8 bits
-bgen-compression zstd	Use zstd algorithm for BGEN compression
-bgen-omit-sample-identifier-block	Omit the sample identifier block

4.2.3.3 Generate BGEN index

```
bgenix -g pvcf.norm.bgen -index -clobber
```

Command section	Annotation
bgenix	bgenix
-g pvcf.norm.bgen	Specify the input genotype file
-index	Create an index file for the given bgen file
-clobber	bgenix will overwrite existing index file if it exists