WES / WGS Pipeline Documentation

This documentation is designed to allow you to set up and run the WES/WGS pipeline either on your own computer (instructions assume a Linux host) or on a Google Compute Engine virtual machine instance. The pipeline is packaged as a Docker container so that setup and configuration is minimized.

Setting up a Google Compute Engine instance

(If you are using your own host machine for running the pipeline, skip this section.) First, if sign up for a Google Cloud account to use Compute Engine (https://cloud.google.com/compute/ (https://cloud.google.com/compute/)). Once you have logged in to your console:

1. Click the menu button (three horizontal bars at the upper left) and select Compute Engine.
2. Click "Create Instance". You might want to read about the options that are available here: https://cloud.google.com/compute/docs/instances/ (https://cloud.google.com/compute/docs/instances/)
3. Choose your machine options (most of these can be changed later); the operating system is probably the most important part, and any Linux distribution (Ubuntu or CentOS) will work fine.
    - You will most likely want a larger primary disk than the default 10G. 20G should be plenty (you can resize this later if you need to).
4. You will need additional storage for reference data and working data. The simplest option to get started is just to create one large Standard disk (1 to 1.5TB). You can resize it later if needed.
    - To do that now, click on " Management, disk, networking, SSH keys", then click on the "Disks" tab that becomes visible.
    - Under "Additional Disks", click on "Add Item", choose "Standard persistent disk", and set the disk name and size.
5. Notice the options to choose number of virtual CPUs. This will be handy later when you are using the pipeline. For now you can save money by leaving that value small 1 CPUs is plenty for most setup and maintenance.
6. In order to manage your new instance, you will want to install the `gcloud` management tool on your local machine. See: https://cloud.google.com/sdk/gcloud/ (https://cloud.google.com/sdk/gcloud/)
7. Start your new instance.
8. Create an SSH connection with the instance either from the web browser (click the "SSH" button) or locally using `gcloud compute ssh your-vm-instance` . You can now manage the instance.
9. You need to transfer files to the VM. You can use `gcloud compute copy-files` , `scp` , or `rsync` to do this (see https://cloud.google.com/compute/docs/instances/transfer-files (https://cloud.google.com/compute/docs/instances/transfer-files) and https://cloud.google.com/sdk/gcloud/reference/compute/copy-files (https://cloud.google.com/sdk/gcloud/reference/compute/copy-files)).
10. Copy either the data bundle small (.tar.gz) for the Dockerfile version of the pipeline or the large image bundle (.tar.gz) to the server, then continue with the configuration instructions below.

Configuring the Pipeline Docker Container

On your host machine, make sure you have Docker (https://docker.com (https://docker.com)) installed. You can configure the pipeline in either of two ways:

1. Import from a pre-built archive of the image (as a large .tar.gz file):
Read the documentation here: https://docs.docker.com/engine/reference/commandline/import/ (https://docs.docker.com/engine/reference/commandline/import/)
2. Build the image from the Dockerfile in the data directory.
Read the documentation here: https://docs.docker.com/engine/reference/builder/ (https://docs.docker.com/engine/reference/builder/)
To do this, you simply need to run `docker build -t YOUR-TAG .` in the same directory as the Dockerfile (where `YOUR-TAG` is replaced with a tag you want to use to identify your image, such as `wes_wgs_pipeline`). Building the image this way will require a stable internet connection and several hours.

If you chose to use the pre-built image, you will want to copy the “pipeline-start.sh” file to a conveient location (the “home” directory on a Google Compute Engine VM, for example). This script (discussed in a later section) will make starting the Docker-based pipeline much easier.

Preparing Data Directories

In order to take advantage of default parameters and do the least amount of configuration editing, it helps to start with a directory structure on the host machine that looks similar to this:

```
[+]    working_dir/
[+]        configs/
[f]            WES_config.groovy  (optional)
[f]            WGS_config.groovy  (optional)
[f]            bpipe.config       (optional)
[+]            pipeline/          (optional)
[ ]                [...]  (updated pipeline control scripts)
[+]        fastq/
[+]            normal/
[ ]                [...]  (normal fastq.gz files)
[+]            tumor/
[ ]                [...]  (tumor fastq.gz files)
[+]        tmp/
[f]        bpipe.config  (optional)
[+]    reference_dir/
[+]        hg19/
[f]            ucsc.hg19.fasta
[f]            ucsc.hg19.dict
[ ]            [...]  (other reference files)
[+]            oncotator/
[+]                oncotator_v1_ds_Jan262014/
[ ]                    [...]  (many files)
[f]                tx_exact_uniprot_matches.AKT1_CRLF2_FGFR1.txt  (or other)
[+]            snpEff_data/
[ ]                [...]  (many files)
[+]        mouse_GRCm38/
[f]            genome.fa
[f]            genome.dict
[ ]            [...]  (other reference files)
[+]            snpEff_data/
[ ]                [...]  (many files)
```

When you launch the pipeline’s Docker container instance the “working” (`working_dir`) and “reference” (`reference_dir`) directories should be mounted into the container at specific filessystem paths (`working_dir ==>`

`/Results` and `reference_dir ==> /ReferenceData`). Many pipeline tools depend upon this configuration, particularly that the *reference directory* contains the " `hg19` " subdirectory, and that within that is the reference sequence FASTA file, known SNP locations, annotation databases, etc. If you maintain this directory structure, most of the default configuration should "just work".

The *working directory* should contain the `configs` directory for custom configuration files, along with a directory containing the input data for the pipeline, and a `tmp` directory that will be used by several tools for transient storage while running.

For tumor/normal runs, the pipeline needs some way to identify the difference between "tumor" inputs and "normal" inputs. By default, simply having the word "tumor" anywhere in the input file's path marks it as a tumor sample, and having the word "normal" in the path marks a file as a normal sample. The easiest way to organize files to satisfy this requirement is to have the `tumor` and `normal` subfolders of your data directory as shown (although a custom filename scheme would work as well).

Additionally the pipeline (by default) expects paired-read data, with a fastq file for each "end" of the paired read. The naming convention that the pipeline expects by default is that the two paired fastq files will contain " `_R1` " and " `_R2` ", respectively, in their filenames. So the following pair of files would be recognized:
"SAMPLE001_L001_R1.fastq.gz" and "SAMPLE001_L001_R2.fastq.gz"
Note that the two names are identical except for the "pair tags" `_R1` and `_R2` . If your file pairs are identified differently, you will need to modify "WES_config.groovy" or "WGS_config.groovy" to indicate the correct pattern.

Launching the Docker Container

You could manually run the docker container and mount your i.e. `working_dir` and `reference_dir` into the container at " `/Results` " and " `/ReferenceData` " respectively, but there is a bash script `pipeline-start.sh` provided to help with this. The first time you run the script it will ask you to enter the path of your *working directory* (i.e. `working_dir` shown above) and *reference directory* (i.e. `reference_dir`). It will remember these values (a file `.pipeline-launch-config` is created in the current directory). If you need to change the values in the future, run `pipeline-start.sh` with the `-c` option (for "configure").

The script will now launch the pipeline for you automatically, and you should see the container's bash prompt. Your current directory is " `/Results` " in the container, which corresponds to your *working directory* on the host machine.

If you want to launch the container manually, your command will look similar to the following:

```
sudo docker run  -v /mnt/data_disk/reference_dir:/ReferenceData \
                 -v /mnt/data_disk/working_dir:/Results \
                 -ti wes_wgs_pipeline
```

Where we assume that the *reference directory* on the host machine is located at `/mnt/data_disk/reference_dir` and the *working directory* on host is located at `/mnt/data_disk/reference_dir` and that `wes_wgs_pipeline` is the container image's *tag*. You can also use a Container ID (shown by the `sudo docker images` command) in place of the tag.

Adjusting Configration

You can get the pipeline's default configuration using the tool `get-pipeline-config` in the running container. It will create the `configs` directory if necessary and retrieve the `WES_config.groovy` and `WGS_config.groovy` files. You can edit those to suit your data and when you run the pipeline using the tool they will override the defaults. To do this manually, copy them back to the `/pipeline` directory when you are finished editing.

If you want to explore adding or modifying Bpipe entry-point scripts, run `get-pipeline-config -s` to also retrieve the full set of entrypoint scripts into `configs/pipeline/` .

To configure the pipeline, simply make edits in either "WES_config.groovy" or "WGS_config.groovy".

One configuration option that you will almost always need to configure is the `INTERVALS` parameter for the WES pipeline mode. This parameter must be set to the file path of the `.bed` file defining the exome targets for your sequencing run. For convenience, the default is set to look for a subdirectory called `wes_bed` inside the *working directory* , and use a file named `regions.bed` from that subdirecotry (path = " `wes_bed/regions.bed` "). If you place and name your `.bed` file appropriately, you will not need to configure this.

As another example, if your fastq filenames looked like "SAMPLE001_L1_1.fastq.gz" and "SAMPLE001_L1_1.fastq.gz" and you were performing a Whole Exome analysis, you would modify "WES_config.groovy" and change the line defining the `PAIR_PATTERN` to the following:

```
PAIR_PATTERN="%_*.fastq.gz"
```

Note the use of the *wildcards* `%` and `*` . In Bpipe, the `%` wildcard in a file name pattern tells Bpipe that is is OK to *parallelize* across that portion of the name. The `*` tell Bpipe that it should allow any values at that location, but it should *group* those files (in sorted order) and send them through the same parallel branch. So, the pattern `"%_*.fastq.gz"` tells Bpipe that the last thing following an underscore and before the file extension is important for grouping (the pairs), and that everything prior to that, if different, may be used for parallelization. The pipeline's default pair pattern is `"%_R*.fastq.gz"` , which means that input split-read fastq files will be grouped by the `_R1` and `_R2` preceeding the extension, and that all other differences before that point may be used to parallelize.

Applying the configuration

After you review the configuration parameters and edit the options that require changes, the next time you execute `run-pipeline` , the configurations will be applied. If you are not using `run-pipeline` , but are running Bpipe manually, you will need to copy the appropriate configuration file into the `/pipeline/` directory prior to running.

Running the pipeline

Once the data is in the correct locations and you are ready to run, you can use the "helper script" `run-pipeline.sh` (in the pipeline files bundle, and installed as `run-pipeline` on the Docker version). The helper script makes starting Bpipe simpler by providing common defaults.

Helper script/command: (`run-pipeline`)

The helper script will start Bpipe with the correct entry point and arguments
automatically, given `WES` or `WGS` mode and (for some commands) files to operate on. Most commands will use

defaults for the file lists (indicated by the […] brackets below).

In addition, the helper script will automatically use any config files you place in the “ `configs` ” subdirectory (as shown above), and will update the toolchain with any updated pipeline scripts from the “ `configs/pipeline/` ” subdirectory automatically.

If you are planning to run Bpipe manually, you can still perform only this auto-update step by running `run-pipeline -c` (where `-c` means “configure”).

```
Syntax:

    run-pipeline [-c] [OPTIONS]  MODE [SPECIES] COMMAND  args...

        -c          Copy configurations into place and stop (do not run Bpipe)

        OPTIONS     are Bpipe-specific options such as "-n 4" See Bpipe
                    documentation: http://docs.bpipe.org/Commands/run/

        MODE        is one of:  {WES, WGS}

        SPECIES     is one of:  {human, mouse} where the default is "human"
                    (so human runs may omit this optional parameter)

        COMMAND     is one of (shown with corresponding values for "args...",
                    [...] indicates optional values):

            fastqc                  [tumor-files.fastq.gz normal-files.fastq.gz]
            trim                    [tumor-files.fastq.gz normal-files.fastq.gz]
            full                    [tumor-files.fastq.gz normal-files.fastq.gz]
            full-realign            [tumor-files.fastq.gz normal-files.fastq.gz]
            to-bam                  [tumor-files.fastq.gz normal-files.fastq.gz]
            to-bam-realign          [tumor-files.fastq.gz normal-files.fastq.gz]
            realign                 [tumor.bam normal.bam]
            realign-resume-from-bam  [tumor.bam normal.bam]
            resume-from-bam         [tumor.bam normal.bam]
            oncotator               variants.vcf
            snpeff                  variants.vcf
```

Command descriptions:

`fastqc` : Only runs FastQC on the fastq files, which may be provided, or will default.
`trim` : Runs Trimmomatic on the fastq files, which may be provided, or will default.
`full` : Run full pipeline.
`full-realign` : Run full pipeline and perform realignment around indels.
`to-bam` : Process from fastq to BAM, then stop.
`to-bam-realign` : Process from fastq to BAM and perform realignment, then stop.
`realign` : Performs realignment on pre-existing BAM files.
`realign-resume-from-bam` : Performs realignment on pre-existing BAM files, then continues the pipeline.
`resume-from-bam` : Resumes the pipeline from pre-existing BAM files.
`oncotator` : Produces annotations with Oncotator given a VCF file.
`snpeff` : Produces annotations with SnpEff given a VCF file.

To run Bpipe directly:

Common Bpipe command usage:

```
    bpipe  ENTRYPOINT.groovy  files-pattern

    Where  files-pattern  is often something like
    "/path/to/tumor/*.fastq.gz  /path/to/normal/*.fastq.gz"

    And ENTRYPOINT.groovy is a Bpipe script such as the ones listed below.
```

Bpipe Entry points:

WES

Full Run

`WES_pipeline.groovy` : runs full pipeline, requires tumor and normal file patterns as shown above

`WES_pipeline-realign.groovy` : runs full pipeline with realignment around indels, requires tumor and normal file patterns as shown above

Incremental Running

`WES_fastqc.groovy` : runs only fastqc, requires tumor and normal file patterns as shown above

`WES_trim.groovy` : runs Trimmomatic, requires tumor and normal file patterns as shown above

`WES_process-to-bam.groovy` : runs pipeline until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WES_process-to-bam-realign.groovy` : runs pipeline with realignment around indels until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WES_realign.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels, QC and stopping. Requires tumor and normal BAM file names following script
(i.e.
`"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WES_realign-resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels before continuing. Requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WES_resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

Annotation

`WES_oncotator.groovy` : runs Oncotator on a variant call file (VCF). Requires VCF file name following script.

`WES_snpeff.groovy` : runs SnpEff on a variant call file (VCF). Requires VCF file name following script.

WGS

Full Run

`WGS_pipeline.groovy` : runs full pipeline, requires tumor and normal file patterns as shown above

`WGS_pipeline-realign.groovy` : runs full pipeline with realignment around indels, requires tumor and normal file patterns as shown above

Incremental Running

`WGS_fastqc.groovy` : runs only fastqc, requires tumor and normal file patterns as shown above

`WGS_trim.groovy` : runs Trimmomatic, requires tumor and normal file patterns as shown above

`WGS_process-to-bam.groovy` : runs pipeline until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WGS_process-to-bam-realign.groovy` : runs pipeline with realignment around indels until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WGS_realign.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels, QC and stopping. Requires tumor and normal BAM file names following script
(i.e.
`"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WGS_realign-resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels before continuing. Requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WGS_resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

Annotation

`WGS_oncotator.groovy` : runs Oncotator on a variant call file (VCF). Requires VCF file name following script.

`WGS_snpeff.groovy` : runs SnpEff on a variant call file (VCF). Requires VCF file name following script.

WES mouse

Full Run

`WES_mouse_pipeline.groovy` : runs full pipeline, requires tumor and normal file patterns as shown above

`WES_mouse_pipeline-realign.groovy` : runs full pipeline with realignment around indels, requires tumor and normal file patterns as shown above

Incremental Running

`WES_mouse_fastqc.groovy` : runs only fastqc, requires tumor and normal file patterns as shown above

`WES_mouse_trim.groovy` : runs Trimmomatic, requires tumor and normal file patterns as shown above

`WES_mouse_process-to-bam.groovy` : runs pipeline until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WES_mouse_process-to-bam-realign.groovy` : runs pipeline with realignment around indels until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WES_mouse_realign.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels, QC and stopping. Requires tumor and normal BAM file names following script
(i.e.
`"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WES_mouse_realign-resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels before continuing. Requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WES_mouse_resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

Annotation

`WES_mouse_snpeff.groovy` : runs SnpEff on a variant call file (VCF). Requires VCF file name following script.

WGS mouse

Full Run

`WGS_mouse_pipeline.groovy` : runs full pipeline, requires tumor and normal file patterns as shown above

`WGS_mouse_pipeline-realign.groovy` : runs full pipeline with realignment around indels, requires tumor and normal file patterns as shown above

Incremental Running

`WGS_mouse_fastqc.groovy` : runs only fastqc, requires tumor and normal file patterns as shown above

`WGS_mouse_trim.groovy` : runs Trimmomatic, requires tumor and normal file patterns as shown above

`WGS_mouse_process-to-bam.groovy` : runs pipeline until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WGS_mouse_process-to-bam-realign.groovy` : runs pipeline with realignment around indels until combined tumor/normal BAMs are produced (plus QC) then stops, requires tumor and normal file patterns as shown above

`WGS_mouse_realign.groovy` : runs pipeline starting from tumor/normal BAM files, performing realignment around indels, QC and stopping. Requires tumor and normal BAM file names following script
(i.e.
`"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WGS_mouse_realign-resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, performing

realignment around indels before continuing. Requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

`WGS_mouse_resume-from-bam.groovy` : runs pipeline starting from tumor/normal BAM files, requires tumor and normal BAM file names following script
(i.e. `"merged_bam/tumor.bam merged_bam/normal.bam"`)

Annotation

`WGS_mouse_snpeff.groovy` : runs SnpEff on a variant call file (VCF). Requires VCF file name following script.

Pipeline Outputs

The pipeline will place its outputs in the *working directory* you used when starting the pipeline. Within the pipeline container this directory is at the path " `/Results` ", but it may be anywhere you like on your host system. On a Google Cloud VM, you almost certainly want this directory to live on a large persistent disk device (that isn't your primary "boot" drive). Assuming you started with a directory structure as shown in the "Preparing Data Directories" section, the resulting directory structure will look like this:

```
[+]    working_dir/                (mapped to /Results in Docker)
[-]        .bpipe/
[-]        bam_qc/
[-]        configs/
[-]        fastq/
[-]        mapped_reads/
[-]        merged_bam/
[-]        oncotator/
[-]        reads_qc/
[-]        snpeff/
[-]        tmp/
[-]        variant_calls/
[f]        bpipe.config  (optional)
[f]        commandlog.txt
[f]        jacquard.log
[f]        strelka_config.ini
```

Variant Calls

The `variant_calls` subdirectory contains the resulting variant call files (VCFs) for the pipeline run. The WGS version is shown expanded below; the WGS version would be lacking the `sv` subdirectory.

```
[+]    variant_calls/
[-]        consensus/
[-]        MuTect2/
[+]        Strelka/
[-]            chromosomes/
[-]            config/
[-]            results/
[+]        sv/
[-]            breakdancer/
[-]            lumpy/
[-]            consensus/
[-]            pindel/
```

Most of the directories contain the `.vcf` files (and sometimes additonal information) directly. The top-level `consensus` directory contains consensus combinations of the calls from all participating callers. Strelka places its calls in the `results` subdirectory. The `sv` (Structural Variants) subdirectory in the WGS pipeline result contains results from the SV callers. Breakdancer does not produce VCF files directly, but its results are combined with the results from Pindel in the `pindel` directory. The `sv/consensus` directory contains SV-only consensus combinations.

Annotations

The `oncotator` and `snpeff` directories contain annotation output from Oncotator and Pindel, respectively. By default the consensus UNION is fed into the annotation stage. You can run annotation only on any VCF of interest using the appropriate Bpipe entry point or the `run-pipeline` helper command.

QC Information

QC data is collected at several points during the pipeline run. For read QC, see the `reads_qc` direcory (FastQC). QC on the processed (BAM) alignments is in the `bam_qc` directory. The variant callers each produce different QC reports, and this data is available in the respective caller's output subdirectory.

Storage

After the run, you might want to save space in your long-term storage archive. It is recommended that you keep the fastq files, the merged BAM files, and the variant calls (and annotations). You can remove the `mapped_reads` directory (containing intermediate SAM and BAM files), and the `tmp` and `.bpipe` directories may be removed at any time following run completion. If you would like to keep a log of the run, you can use Bpipe to retrieve the log data (`bpipe log` , see http://bpipe-test-documentation.readthedocs.io/en/latest/Commands/log/ (http://bpipe-test-documentation.readthedocs.io/en/latest/Commands/log/)), but this must be done prior to deleting `.bpipe` .