Eichler Lab Wiki

WGAC: Whole genome assembly comparison (the most recent, Aug 2020, David Gordon)

WGAC is described in the paper by Bailey et al. 2001. However, I (DG) don't find it much use for running the pipeline. Instead, read this:

1. make an empty directory (make sure quota allows you ~300GB since mouse used 276G and chm13t2t used 151GB and wasn't quite finished)
2. git clone git@github.com:EichlerLab/WGAC.git .
3. make an fofn of the sequences making up the reference that you are going to try to find seg dups in and call it fastawhole.fofn
4. add a file chromosomes.txt that contains the names of each chromosome, one on a line
5. I believe (Jan 2020, DG) that the sequence must already be soft repeatmasked. The pipeline does not run repeatmasker but it uses lowercase to make a file of repeats. I've added trf hard masking after fuguizing in middle of pipeline because if the sequence contains lots of tandem repeats such as GACCCGACCCGACCCGACCC… blast will crash and lastz will take days instead of minutes.
6. type: make -f makefile_wgac -k -j 3 >makefile_wgac.out 2>&1
7. when this has successfully completed, the last section (making plots), must be run with DISPLAY set to an X11 xterm. Run: make -f makefile_wgac all_plots

I wish I could say that you can come back in a few days to a week and it would be done, but it never works that way. It always stops due to some problem (usually a temporary cluster problem) and you have to clean up the uncompleted step it was on and restart it. Thus you need to know a little of what it is doing. Hence the next section.

Here is one place you might not make it through without manual intervention:

see this stage in makefile_wgac:

```
global_align_110k_done: made_sge_align_batch_110k
    ./run_global_align_110k.snake.sh
```

This usually fails. (as of May 2021, DG) It is trying to align 2 sequences that did align before repeats were added back. After repeats are added back (a process known as "defuguization") the sequences might be so different that they can't align. If any pair of sequences can't be aligned, then the snake job will fail so the flag global_align_110k_done will not be set.

Determine how many failed as follows:

```
wc -l sge_align_batch_110k.sh
```

gives the number attempted with special parameters.

```
ls global_align_110k_flags | wc -l
```

gives the number of alignments with special parameters that succeeded. The difference gives the number that failed.

If a large fraction (e.g., half) of your alignments failed, something is wrong. For example, in chimpanzee there were 137,508 total alignments, but 150 could not be aligned (0.1%). In the case of bonobo, there were 250 (0.8%) that failed out of the 33,132 alignments to be done as shown in sge_align_batch.sh In the case of marmoset 2233 failed out of a total of 240673 or about 1%

To help determine what failed, run:

```
./summarize_global_align_errors.py
```

and look at the output file: summarize_global_align_errors.txt You might look at some of the alignments themselves to see if you think they are valid segmental duplications

If you have convinced yourself that there is nothing wrong, then push the program past this step, making it ignore the failed alignments by typing:

```
touch global_align_110k_done
```

and restart the

```
make -f makefile_wgac -j 3 >makefile_wgac2.out 2>&1
```

and allow WGAC to complete. (DG, May 2020)

Other Possible Problems

I've (DG) gotten this error:

Error recording metadata for finished job ([Errno 2] No such file or directory: 'fugu_trf/chr2_274.fugu'). Please ensure write permissions for the directory /net/eichler/vol27/projects/hprc/nobackups/chm13v2_wgac3/.snakemake

which is nonsense: the file existed and .snakemake had write permissions. It was only necessary to restart the pipeline for it to complete. I've get this error occasionally but usually not.

What the makefile_wgac is doing

Warning: some of this is out of date due to changes in the cluster such as queue names, locations of files, moving programs from people's home directories into the git repository, etc. These changes are constantly occurring and we can't spend all our time updating the wiki.

Mask sequence(s)

<WRAP center round important 70%> If your assembly has already been repeat masked by UCSC , it is advised to use the rmsk.out file to mask your sequences especially for well annotated references like human and mouse. This has been the default practice for WGAC analysis. For pacbio assemblies in contigs , we ignore simple-repeat rich contigs ⇐ 20 kbp for the self-blast step.

For primates, where the repeat masker library is incomplete, you might get lot of hits from the fugu blast step representing transposons etc, masking with mammalian library in addition to primate specific repeats and increasing the seed size to 500 bp will help reduce the total blast hits significantly. Most of the unmasked repeats will fall in the size range of 250-500 bp. </WRAP>

If RepeatMasker output is already available for your sequences (as it commonly is for species assemblies from UCSC), it is simpler to mask your sequences before splitting them into 400 Kbp segments. Then, after splitting, you can recreate .out files for each segment using the lowercased letters.

Create "fuguized" sequences--What the makefile is doing

First WGAC checks that you have repeatmasked the sequences (softmasked–make repeats lowercase). See ~dgordon/pipelines/repeatmasker_whole for a repeatmasker pipeline

Break (fractionate) the FASTA records into smaller pieces. The "-s" flag specifies size in base pairs for each piece.

I've found that fasta_fractionate fails (silently) if the line length is very long, perhaps thousands of bases on a line. The effect is to have files that look like this (headers with no bases):

>test.fasta_002
>000001F_1_57587592_quiver

and other files that have more bases than -s specifies. Thus I (DG, 6/2016) have it make shorter lines and put the output on /tmp. Then WGAC takes these fasta files and splits them into fasta files that have only one sequence per file, writing these fasta files into fastawholesplit.

I have fasta_fractionate read the resulting shorter lines from fastawholesplit and write the output to fasta.

```
mkdir fasta
fasta_fractionate -f fastawhole -s 400000 -o fasta
```

WGAC then reads the (small) files in fasta and looks for lowercase, making fake repeatmasker .out files like this:

```
	mkdir -p mask_out
	ls fasta |xargs -i perl ./maskOutGenFromLowCase.pl fasta/{} mask_out/{}.out
```

a symlink is created to the mask_out directory

```
ln -s Masking/step1/mask_out/ .
```

Fugu-like sequences (without repeats) are created reading the fasta directory and the fake RepeatMasker .out files. RepeatMasker files should be 1-based. For the human genome this step takes 13 minutes and fugu directory should be about 1.6 Gb. The results are put into fugu.

```
mkdir fugu
fasta_fuguize_batch.pl -f fasta -r mask_out -o fugu
```

trf is then run on the fugu-ized sequences. This is because long tandem repeats will cause blast to crash and lastz to take forever (a week?). trf is run by creating a directory "trf" which has links to the files in the "fugu" directory. The trf-masked sequences are put into fugu_trf

BLAST fuguized sequences

Make a BLAST database from a single huge FASTA file.

```
mkdir blastdb
dir_cat fugu blastdb/bofugu
formatdb -i blastdb/bofugu -o F -p F -a F
rm -f blastdb/bofugu
```

Run BLAST on the fuguized sequences. Run this step concurrently with the following self-blast step.

The SGE BLAST script takes the following arguments:

- -b BlastDB source path (default: blastdb/bofugu)
- -t BlastDB temporary directory (default: /var/tmp/blastdb)
- -i input directory (default: fugu)
- -o output directory (default: blastout)

If you are in the WGAC working directory, the only argument you should need to submit is the temporary directory path. This path should be based on your username so you can easily cleanup after yourself later.

```
blastout_done : blastdb/bofugu.nsq
	mkdir -p blastout
	qsub -pe orte 100 -N wgac_blast_$(TRIM_SPECIES) -sync y ./blast64.sh -t /var/tmp/$(shell whoami)/$(TRIM_SPECIES)/blastdb
	./checkIfBlastSucceeded.sh
	touch blastout_done
```

(On panTro3 dataset this took about 6 hours. DG 6/2016)

################ for anyone working on this pipeline ##############################

qsub -orte in the current version of the sge software has a bug in it. If 10 jobs land on the same node, they will each be limited to 10% cpu. So this step will not be completely parallelized and may take a long time. I'm trying (DG 4/21/2020) to get this fixed. But not so far…

The alternative solution would be to rewrite this in snakemake. Here is the difficulty with that: blastn is fastest (at least the original writers of this pipeline thought so) when the blast database is on a local disk. So the qsub -orte method first copies that database to each of the nodes that will be running blastn. it can do that because sge allocates the nodes in advance of running, and those nodes are kept until the entire blast step completes. snakemake, on the other hand, will use nodeA for one job and when nodeA completes, it might get nodeA again or it might get nodeB. So it needs to be prepared to run on a node that doesn't yet have the blast database copied. It also needs to handle the case in which 2 jobs land on the same node that doesn't yet have the database and they must not

both try to copy the database. One solution is to have a cluster-wide (or node local) lock. I have found that making a write lock on a file on the network drive will work–it will cause only the first job to get the lock and subsequent jobs will be suspended. This is what you want. When the first job releases the lock, subsequent ones will find that the database is copied and won't need to do it. This could be done by means of a dummy file on the local disk or a dummy file that includes the node name on the network disk. I did something like this in python for falcon in the 2015 incarnation.

################ end of notes for people working on this pipeline ################

When BLAST is complete, parse the output.

First check for files of nohit based on the size and remove these files. File size will be less than 1300 (containing only a header and message no hits found).

This command will produce nonsense output if $size is empty and therefore still less than 1300.

```
	perl ./rmNohitfiles.pl blastout
```

Check for files with problem aligns (e.g., files without blast output message Matrix: blastn matrix:30 -80). Sometimes the blast output file is truncated at the end. In this case you have to either fix the file by adding summary to the end of the file or rerun the blast for the specific file (this latter step is probably easier).

```
find blastout -type f -exec grep -L 'Matrix' {} \;
```

(This can take just a few minutes or several hours, depending on the dataset. DG, 6/2016)

Parse files with a seed size of 250 bp and identity 88%. You can change the parameters, seed size and identity. Sometimes if the repeats are not removed well, such as with elephant and zebra finch, you may want to use a seed of 500. This step disregards self hits and outputs data/bo_self.parse. <p>

The below command took around 15 to 45 minutes for chimpanzee (depending on the dataset). If a perl module was loaded, it failed, but after module unload perl, it worked (it uses /usr/bin/perl) (DG, 6/2016, 5/2017)

```
mkdir -p data

blastparser42.pl -in ./blastout \
  -out data/bo_self.parse \
  -noprevq -v \
  -output '-ssort=>name, -hsort=>qb' \
  -filter '-min_bpalign=>250, -min_fracbpmatch=>0.88, -max_%gap =>40, -no_subject_self => yes, -no_doubleoverlap=>score'
```

Lastz compares each chunk to itself

This is run using a snakefile lastz_self.snake

Replace the letters “RYKMSWBDHV” with “N” and switch all letters to uppercase.

```
cp -r fugu fugu2
fasta_only_ATCGN.pl -p fugu2 -o fugu2 -u
```

(The cp -r step took several hours on one dataset. The fasta_only_… also took about several hours. However, on hg19 on vol26 (with gpfs) it just took a few minutes. Perhaps it all depends on the speed of IO rather than the dataset. DG 6/2016, 5/2017)

Lastz Internals

Read this only if you have troubles with running lastz following the instructions above and need to understand what is going on internally. (Be aware that originally there was a program called “webb_self” or “selfblast” which only ran on a Sun. I (DG, Nov 2019) replaced it with lastz.

Run lastz. The directory “selfblast” will contain one output file per fugu file.

Parse self-blast output.

```
mkdir tmp
blast_lav_break_self_overlap2.pl --in selfblast --out tmp

mkdir -p data
blast_lav_hit_by_hit.pl --in tmp --out data/lav_int2.parse -options 'MIN_BPALIGN=>200, MIN_FRACBPMATCH=>0.88, MAX_%GAP => 40, SKIP_SELF => 0, SKIP_OVERLAP=>1'

rm -rf tmp
```

The above steps took 9 hours for chimp. (DG, 1/2017)

Merge parsed BLAST and parsed LASTZ results

The following steps combines the results of inter-block (blast) with intra-block (lastz):

```
data/both.parse : data/inter_block.parse data/intra_block.parse
	cp $< $@
	sed 1d $(lastword $^) >> $@
```

Defuguize regions

Defuguize a pairwise blast table (both.parse) using the original “fuguized” sequence and the RepeatMasker output. This creates a file named “both.parse.defugu” using subdirectories “fugu” and “mask_out”.

This step may take 9 hours, depending on size of “both.parse”. (Hmmm…it took me about 30-45 minutes on hg19 and my results matched Archana's. -DG June, 2016 and June 2017)

```
blast_defuguize_hit_by_hit.pl -t data/both.parse -d .
```

Trim ends

Trim sequence ends.

```
cd data
mkdir -p step_8_mpi/defugu
```

Split the defugu file into smaller files and store those files in a new directory called “newdir”.

The first argument is the name of the defugu file. The second argument is the total number of smaller files you want to have. The third argument is the number of lines in defugu file.

```
data/step_8_mpi/defugu : data/both.parse.defugu
	mkdir -p $@
	perl ./split.pl $< 300 $(shell wc -l $<)
	find newdir/ -type f -exec cp {} $@ \;
	rm -rf newdir
```

Submit the end trimming job. If you are in the root WGAC directory (parent of “data”), you can run the below command with “species_name” .

```
trim_ends_done : data/step_8_mpi/defugu
	mkdir -p data/step_8_mpi/trim
	./make_trim_jobs.sh ${TRIM_SPECIES} >trim-ends_${TRIM_SPECIES}.sh
	chmod +x trim-ends_${TRIM_SPECIES}.sh
	qsub -N trim-ends_${TRIM_SPECIES} -pe orte 100 -sync y ./sge_commands.sh trim-ends_${TRIM_SP
ECIES}.sh
	./checkIfTrimEndsSucceeded.sh
	touch trim_ends_done
```

Collect the generated trim files into one file named ParallelOutput.trim and located in data/step_8_mpi/trim.

```
data/both.parse.defugu.trim : trim_ends_done
	perl ./collectTrim.pl data/step_8_mpi/trim
	mv data/step_8_mpi/trim/ParallelOutput.trim $@
```

Generate both.parse.defugu.trim.fixed.trim

This step fixes for overlap alignment ('overlaps' in files), reduces the number of lines (one half of one percent), and takes 2 minutes.

```
blast_align_by_align_overlap_fix3.pl -i both.parse.defugu.trim
```

Generate both.parse.defugu.trim.fixed.trim.defrac

This step takes 3 minutes for the human genome. Note that the parameter 400000 was used in an earlier step to fractionate the chromosome files in fastawhole/. Adjust your value here accordingly.

```
blast_defractionate3.pl -s 400000 -t both.parse.defugu.trim.fixed.trim -f ../fastawhole
```

<div important> At this step, check the size of `both.parse.defugu.trim.fixed.trim.defrac`. If this file is larger than 100MB, the global alignments step may not be completable and you may crash the head node.

Run global alignments

- Generate batch script to run the global alignment job with the default alignment length 60000

```
/net/eichler/vol18/araja/wgac/wgacAlign.sh fasta >sge_align_batch.sh
```

(DG: for me this step took more than 2 hours on hg19)

Make sure that the number of lines in the sge_align_batch.sh script matches to the number of lines -1 in the data/both.parse.defugu.trim.fixed.trim.defrac

wgacAlign creates new directories: both_tmp and align_both (one is for synchronization between CPUs, the other holds output files).

- Submit the global alignment job to the cluster. This step might take a day or two depending on the traffic on vol18

```
qsub -N global_align -l h_rt=24:00:00 -pe orte 50 /net/eichler/vol2/local/inhousebin/general_pipe/sge_commands.sh sge_align_batch.sh
```

(note from DG, 7/2016: I found that with panTro3 and 244,000 lines in sge_align_batch.sh, this job silently failed. I was able to get it to complete by breaking sge_align_batch.sh into 2 files, each with 122,000 lines, and submitting each separately. The way you can tell that it is successfully running is that these files should exist and be fairly large:

-rw-r--r-- 1 dgordon eichlerlab 165621920 Jul 7 13:06 global_align.o19765449
-rw-r--r-- 1 dgordon eichlerlab 246274747 Jul 7 13:06 global_align.e19765449

where the #s 19765449 will be replaced by your jobid

and the .o file above will have some lines similar to:

OUTFILE /net/eichler/vol18/dgordon/wgac/panTro3b/data/align_both/0024/both0122012
SUBSEQ1
SUBSEQ2
ALIGN

while if it fails these files will be at most a few hundred bytes and will not contain any of these keywords.)

- When the job finishes,

1. count the number of files in all subdirectories of align_both
2. count the number of lines in the defrac file

```
1. count the number of files in all subdirectories of align_both
ls -d data/align_both/*/ | xargs -i ls -1 {} | wc -l

2. count the number of lines in the defrac file
wc -l data/both.parse.defugu.trim.fixed.trim.defrac
```

For the human genome, Saba got 111726 and 111738, respectively. For this genome, the final count of output files ideally should be 111,737 due to header line of defrac file.

Remove the temporary directory . If everything went correctly with the first alignment step, this directory should be empty already.

```
rm -rf both_tmp/both*
rm -rf both_tmp
```

If the numbers dont match, regenerate the sge_batch script again with an alignment size of 110000 and submit the script to the cluster. (Note from DG 6/2016: I initially got 11 fewer alignments than the size of both.parse.defugu.trim.fixed.trim.defrac. The point of re-running wgacAlign.sh with 110000 is to recover those (11) missing alignments–not to rerun all alignments. I've heard it is written to create sge_align_batch_110k.sh with just the missing alignments. I was able (a slightly different way) to recover those 11 missing alignments.)

```
/net/eichler/vol18/araja/wgac/wgacAlign.sh fasta 110000 >sge_align_batch_110k.sh
qsub -pe orte 2-$(echo "`cat sge_align_batch_110k.sh|wc -l`+1"|bc) /net/eichler/vol2/local/inhousebin/general_pipe/sge_commands.sh sge_align_batch_110k.sh
```

```
ls -d data/align_both/*/ | xargs -i ls -1 {} | wc -l
```

There should be an increase in number of output files. For hg19, this should be close to 111,737.

Compute scores

Compute scores and other statistics (e.g. Kimura's). This step takes 80 minutes for hg19. (It took 9 hours for me to run the align_scorer_batch2.pl step on hg19 and 12 hours on panTro3. It generates a huge amount of output that should be redirected to a file. DG 6/2016)

<div todo> Distribute score calculations to cluster.

Output is a new directory “align_both_data” and a new file “both.alignscorerdata”.

```
cd data
align_scorer_batch2.pl -a align_both \
    -d align_both_data -p both \
    -o both.alignscorerdata
```

Output of the first command (below) minus one (due to the header) should be equal to the output of the second command. For hg19, Saba got 111499 for the second command.

```
wc -l both.alignscorerdata
grep -c 'align_both' both.alignscorerdata
```

Merge files defractionated file with score and stats data. This step takes 1 minute for hg19.

```
table_merger.pl -s'row(\d+):both(\d+)' \
    -i both.parse.defugu.trim.fixed.trim.defrac:both.alignscorerdata \
    -o defrac.align
```

BLAST welder

Run the “welder”. This step takes 1 hour for hg19. The welder produces several files all with the same prefix `oo.weild10kb`.

The -g parameter specifies smallGap:largeGap:overlap. To join two pairwise alignments, both query and subject should have gaps < largeGap and at least one of them should be < smallGap.

```
blast_hit_by_hit_welder2.pl -i defrac.align -g 40:10000:20 -o oo.weild10kb
```

Create two files named blah.blah.cull. This step takes less than 1 minute for hg19. `oo.weld10kb.join.all.cull` will be used in next step. Note from DG, Dec 2021: this step has become an extreme bottleneck, sometimes taking as much as 3 weeks. There are a pair of nested loops in it that go through each line in defrac.align so if that file is, say, twice as large, this program will take 4x as long.

```
table_line_culler2.pl -p oo.weild10kb.pieces.all -t'$c[27]>=0.90 && $c[64]>=1000'
table_line_culler2.pl -p oo.weild10kb.join.all -t'$c[27]>=0.90 && $c[22]>=1000'
```

(Note from DG 6/2016: Archana eliminated from oo.weild10kb.join.all.cull any lines referring to alternate haplotypes. Without doing this, the plots will be different from hers.)

WGAC calculations should be complete now. What follows are steps related to analysis of these data.

Create summary files

Create a “super dup” file. (DG 1/2017): GenomicSuperDup.tab is THE most important file in the WGAC pipeline. It is the product. Each line is a seg dup.

<div todo> Update makeTrack.pl to output the size of otherChrom instead of otherEnd - otherStart for the otherSize field.

```
data/GenomicSuperDup.tab : data/oo.weild10kb.join.all.cull
	perl ./makeTrack.pl $< > $@
```

Make sure there isn't redundancy in the super dup file. (Below script is obsolete now)

(Note from DG 6/2016: Archana eliminated from GenomicSuperDup.tab any lines referring to alternative haplotypes.)

The fields (I think) are the following:

```
1 chromosome
2 start pos
3 end pos
6 orientation _ (underscore) is reverse, + is forward
duplicated region:
7 chromosome
8 start pos
9 end pos
11  this is some ID for seg dups.  if 2 different lines were the same except have 1,2,3 and 7,8,9 reversed, then
they had the same field #11.

(Please add to this table as you discover the meaning of columns.)

(going through the perl script I believe I have found the meaning for the last 13 columns but I am not positive, MV 08/23/2017)
17 The file containing the NW (global) base-level alignment of the 2 seg dups for the line.
18 length of alignment (includes gaps and Ns)
19 indel_N (not sure what the _N and _S are, I think they are indels that are Ns vs real bases)
20 indel_S (but even if I am right I do not know if this is the number of indels or the number of bases that are indels )
21 the number of aligned bases not including Ns and gaps. (i.e. matches + mismatches)
```

```
22 matched bases
23 mismatched bases
24 transitions
25 transversions
26 similarity (not including indels, perfect agreement is 1.0)
27 similarity (including indels)
28 K_jc (Jaccard score)
29 k_kimura (Kimura score)
```

```
perl /net/eichler/vol4/home/jlhudd/ps/wgac/rmRedunInSUperDup.pl GenomicSuperDup.tab
```

Calculate similarity and length distribution statistics. You can use these data to create charts for length distribution (use 5th and 6th columns from `all.join.cull.lenDis`) and similarity distribution (use 5th and 6th columns from `all.join.cull.simDis`). Check out Saba's charts and make similar ones:

- /net/eichler/vol5/home/ssajjadi/Marmaset/Analysis_withChrUn_newWindowMasker/chromOnly/Charts/Stats.xls

```
mkdir stats
perl ~ssajjadi/wgacbin/getSimStat4Cull.pl oo.weild10kb.join.all.cull > stats/all.join.cull.simDis
perl ~ssajjadi/wgacbin/getLenStat4Cull.pl oo.weild10kb.join.all.cull > stats/all.join.cull.lenDis
```

Use the following output files to create more accurate charts.

```
perl ~ssajjadi/wgacbin/getSimStat4Cull-05.pl oo.weild10kb.join.all.cull > stats/all.join.cull.simDis_05
perl ~ssajjadi/wgacbin/getLenStat4Cull-2K.pl oo.weild10kb.join.all.cull > stats/all.join.cull.lenDis_2K
```

The following commands are used to find nr duplication on each chromosome and also inter/intra non-redundant (NR) duplication. Evan usually needs only the total NR duplication which will be the last line in `AllDupLen` file.

```
perl /net/eichler/vol4/home/jlhudd/ps/wgac/oneHitPerLine.pl oo.weild10kb.join.all.cull > stats/oneHitPerLine
```

Sort based on 1st (chromosome name) and 2nd (start) column.

```
sort -k 1,1 -k 2,2n stats/oneHitPerLine > stats/oneHitPerLine_sort
```

The file `AllDupLen` contains the total NR duplication length per chromosome and percentage.

```
cd stats
perl ~ssajjadi/wgacbin/mergeHit.pl oneHitPerLine_sort > mergeHit
```

Generate the nonredundant all duplication lengths ("nr all dup len").

```
perl /net/eichler/vol4/home/jlhudd/ps/wgac/statLenPerChr.pl mergeHit > AllDupLen
```

Generate intrachromosomal and interchromosomal duplication information. The interDupLen and intraDupLen will be used to graph the length distribution for inter and intra.

```
grep "intra" oneHitPerLine_sort > intraSort
grep "inter" oneHitPerLine_sort > interSort

perl ~ssajjadi/wgacbin/mergeHit.pl interSort > interMerge
perl ~ssajjadi/wgacbin/mergeHit.pl intraSort > intraMerge

# nr intra dup len
perl /net/eichler/vol4/home/jlhudd/ps/wgac/statLenPerChr.pl intraMerge > intraDupLen

# nr inter dup len
perl /net/eichler/vol4/home/jlhudd/ps/wgac/statLenPerChr.pl interMerge > interDupLen
```

Generate a non-redundant duplication report.

```
/net/eichler/vol4/home/jlhudd/wgac/nr_report.sh ../../fastalength.log interDupLen intraDupLen AllDupLen nr_stats.tab
```

Generate "all WGAC" report.

```
groupBy -i oneHitPerLine_sort -g 1 -c 5 -o sum > all_all_dup
groupBy -i interSort -g 1 -c 5 -o sum > inter_all_dup
groupBy -i intraSort -g 1 -c 5 -o sum > intra_all_dup
/net/eichler/vol4/home/jlhudd/wgac/nr_report.sh ../../fastalength.log inter_all_dup intra_all_dup all_all_dup all_stats.tab
```

To create "Length vs. Similarity" and "Length vs. k_kimura" charts as in Saba's example spreadsheet, parse out columns 23, 28, and 34 from `oo.weild10kb.join.all.cull`.

```
cut -f 23,28,34 ../oo.weild10kb.join.all.cull > length_similarity_kimura
```

Generate plots for WGAC by duplication length, duplication identity, chromosome, and Kimura's k. Run this script in the `stats` directory.

```
Rscript /net/eichler/vol4/home/jlhudd/pipelines/wgac/plots.R
```

Calculate additional WGAC summary statistics.

```
cat ../GenomicSuperDup.tab \
    | gawk -F '\t' '{ if($26>0.94)  print $1"\t"$2"\t"$3; }' > gt94WGAC.tab

cat ../GenomicSuperDup.tab \
     | gawk -F '\t' '{ if($26<=0.94) print $1"\t"$2"\t"$3; }' > le94WGAC.tab

/net/eichler/vol7/home/ginger/bin/coordsMerger_sort.pl \
    -i gt94WGAC.tab -n 0 -b 1 -e 2 -u -o gt94WGAC.merged

cat gt94WGAC.merged \
    | gawk '{ if($3-$2>=10000) print $0;}' > gt94WGAC_ge10K.tab

cat ../GenomicSuperDup.tab \
    | gawk -F '\t' '{ print $1"\t"$2"\t"$3; }' > allWGAC.tab

/net/eichler/vol7/home/ginger/bin/coordsMerger_sort.pl \
    -i allWGAC.tab -n 0 -b 1 -e 2 -u -o mergedWGAC.tab

/net/eichler/vol7/home/ginger/bin/coordsMerger_sort.pl \
```

```
    -i le94WGAC.tab -n 0 -b 1 -e 2 -u -o le94WGAC.merged

/net/eichler/vol7/home/ginger/bin/twoPartOvp_mgsrt.pl \
    -i le94WGAC.merged -f  -j gt94WGAC.merged -t -L -o le94WGAC_only.tab
```

Visualize WGAC with Parasight

The current way to do this (Dec 2021, DG) is to just run:

```
make -f makefile_wgac plots
```

All of the description below is what this does and what you might have to do if the above runs into a problem. The above command will also generate the plots for length and similarity histograms called:

```
lendis_2K.png
lendis.png
simdis_05.png
simdis.png
```

Generally Evan will also want to have a set of plots for just chromosomes without the unplaced contigs. To do this, start with file data/oo.weild10kb.join.all.cull and extract the lines that only refer to chromosomes. Do this in a separate empty directory. Then create a new set of plots. Here is an example:

```
> cat just_chrs.sh
cat oo.weild10kb.join.all.cull | awk '{if ( $1 ~ "^chr" && $5 ~ "^chr") print }' | grep -v chrUn >oo.weild10kb.join.all.cull.just_chrs

perl ./getSimStat4Cull.pl oo.weild10kb.join.all.cull.just_chrs > all.join.cull.simDis
perl ../getLenStat4Cull.pl oo.weild10kb.join.all.cull.just_chrs > all.join.cull.lenDis
perl ../getLenStat4Cull-2K.pl oo.weild10kb.join.all.cull.just_chrs >all.join.cull.lenDis_2K
perl ./getSimStat4Cull-05.pl oo.weild10kb.join.all.cull.just_chrs >all.join.cull.simDis_05

Rscript ../plots.R
```

Use the statistics generated in the previous steps to visualize WGAC results with Parasight. The main views are called “global”, “general”, and “blowup” views.

Generate `xw.join.all.cul` (column 23 is the number of aligned bases, column 28 is the similarity (matches/aligned bases)).

```
cd data
cut -f 1,2,3,4,5,6,7,8,23,28 oo.weild10kb.join.all.cull > xw.join.all.cull
perl /net/eichler/vol4/home/jlhudd/ps/para/wgac/xwalign.pl xw.join.all.cull > xw.al
```

Before modifying the script, create a hash table for chrom length. Note that `fastalength.log` may be in the parent directory of the `data` directory.

```
perl /net/eichler/vol4/home/araja/Useful-Code/wgac/writeGenomeLengHash.pl ../fastalength.log > length_hash
perl /net/eichler/vol4/home/araja/Useful-Code/wgac/writeGenomeLengtab.pl ../fastalength.log > length_tab
```

Calculate intersection between WGAC and WSSD.

```
/net/eichler/vol4/home/jlhudd/bin/chrom_cut.pl /path/to/wssd/chrom.out \
    stats/gt94WGAC_ge10K.tab \
    /path/to/wssd/wssdGE10K_nogap.tab \
    gt94WGAC_ge10K \
    WSSD \
    shared \
    gt94WGAC_ge10K_WGAConly \
    WSSDonly \
    stats/mergedWGAC.tab \
    allWGAC \
    stats/le94WGAC_only.tab \
    le94WGAC > wgacCMPwssd.out
```

Return to the root WGAC directory.

```
cd ..
```

Global views

Create a list of sequence lengths for Parasight. Delete random and unplaced chromosomes from `showseq.out` and the “total” line.

```
sed '/random/d;/chrUn/d' data/length_tab > showseq.out
```

Manually correct the order of the entries in `showseq.out` if necessary.

Prepare output directories.

```
mkdir -p globalView
cd globalView
```

Run the Parasight command for 10K_90, 10K_94, 10K_97,5k_90, 5k_94, 5k_97. Here we are running it for 5k_90. Each run of the command generates a file named like `parasight_5k_90.ps`.

The value of option “printer_page_orientation” formats the result in landscape as opposed to portrait mode.

Generate global views for segments > 5 Kbp and identity > 90%.

```
parasight751.pl -showseq  ../showseq.out -align ../data/xw.al \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst \
    -option '-filterpre2_min=>5000, -filter2_col=>16, -filter2_min=>0.90, -extra_label_on=>0, -seq_tick_label_fontsize => 10, -seq_label_fontsize => 10, -printer_page_orientati
    -precode "&fitlongestline; &print_all(0,'parasight_5k_90');" -die;
```

Repeat for identity > 95%.

```
parasight751.pl -showseq  ../showseq.out -align ../data/xw.al \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst \
    -option '-filterpre2_min=>5000, -filter2_col=>16, -filter2_min=>0.95, -extra_label_on=>0, -seq_tick_label_fontsize => 10, -seq_label_fontsize => 10, -printer_page_orientati
    -precode "&fitlongestline; &print_all(0,'parasight_5k_95');" -die;
```

Repeat for identity > 98%.

```
parasight751.pl -showseq  ../showseq.out -align ../data/xw.al \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst \
    -option '-filterpre2_min=>5000, -filter2_col=>16, -filter2_min=>0.98, -extra_label_on=>0, -seq_tick_label_fontsize => 10, -seq_label_fontsize => 10, -printer_page_orientatio
    -precode "&fitlongestline; &print_all(0,'parasight_5k_98');" -die;
```

Repeat all three identity filters with a 10 Kbp length filter instead of 5 Kbp.

```
parasight751.pl -showseq  ../showseq.out -align ../data/xw.al \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst \
    -option '-filterpre2_min=>10000, -filter2_col=>16, -filter2_min=>0.90, -extra_label_on=>0, -seq_tick_label_fontsize => 10, -seq_label_fontsize => 10, -printer_page_orientati
    -precode "&fitlongestline; &print_all(0,'parasight_10k_90');" -die;

parasight751.pl -showseq  ../showseq.out -align ../data/xw.al \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst \
    -option '-filterpre2_min=>10000, -filter2_col=>16, -filter2_min=>0.95, -extra_label_on=>0, -seq_tick_label_fontsize => 10, -seq_label_fontsize => 10, -printer_page_orientati
    -precode "&fitlongestline; &print_all(0,'parasight_10k_95');" -die;

parasight751.pl -showseq  ../showseq.out -align ../data/xw.al \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst \
    -option '-filterpre2_min=>10000, -filter2_col=>16, -filter2_min=>0.98, -extra_label_on=>0, -seq_tick_label_fontsize => 10, -seq_label_fontsize => 10, -printer_page_orientati
    -precode "&fitlongestline; &print_all(0,'parasight_10k_98');" -die;
```

Repeat all three identity filters at 20 kbp

```
parasight751.pl -showseq  ../showseq.out -align ../data/xw.al -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst -option '-filterpre2_min=>20000, -filter2_co

parasight751.pl -showseq  ../showseq.out -align ../data/xw.al -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst -option '-filterpre2_min=>20000, -filter2_co

parasight751.pl -showseq  ../showseq.out -align ../data/xw.al -template /net/eichler/vol4/home/jlhudd/paratemplate/globalview10k.pst -option '-filterpre2_min=>20000, -filter2_co
```

Convert PostScript to PDF.

```
gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=global_view_5k.pdf parasight_5k_*.ps
gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=global_view_10k.pdf parasight_10k_*.ps
gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=global_view_20k.pdf parasight_20k_*.ps
```

Return to WGAC root.

```
cd ..
```

General views

Prepare Parasight input files for general views.

```
perl /net/eichler/vol4/home/jlhudd/ps/para/writeNr5kextra.pl data/stats/mergeHit \
    | sed 's/black/red/g' > wgac5k.extra

perl /net/eichler/vol4/home/jlhudd/wgac/writeArrangeUsngLoop.pl showseq.out > arrange.out
```

Prepare general view output directories.

```
mkdir -p generalView
cd generalView
```

Set the canvas_bpwidth based on the longest sequence. You can change the color for wgac bars by editing .extra file. In our example it is red.

```
/net/eichler/vol4/home/jlhudd/parasight/parasight751_4graph.pl -extra ../wgac5k.extra \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/template.pst \
    -showseq ../showseq.out -arrangeseq  file:../arrange.out \
    -options "-seq_tick_bp=>10000, -extra_label_on=>0, -sub_on=>0, -printer_page_orientation=>0" \
    -precode "&fitlongestline; &print_all(0,'general-view');" -die;
```

Create a WSSD extra file for Parasight.

```
perl /net/eichler/vol4/home/jlhudd/ps/para/writeWssdExtra.pl wssdGE10K_nogap.tab > wssd.extra
```

If you have gaps.extra or wssd.extra, you can add them to the image as well. You can change the colors by editing .extra files

```
/net/eichler/vol4/home/jlhudd/parasight/parasight751_4graph.pl -extra ../wgac5k.extra:wssd.extra \
    -template /net/eichler/vol4/home/jlhudd/paratemplate/template.pst \
    -showseq ../showseq.out -arrangeseq file:../arrange.out \
    -options "-seq_tick_bp=>10000, -extra_label_on=>0, -sub_on=>0, -printer_page_orientation=>0" \
    -precode "&fitlongestline; &print_all(0,'general-view-with-wssd');" \
    -die;
```

Convert PostScript to PDF.

```
gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=general_view.pdf *.ps
```

Return to WGAC root.

```
cd ..
```

Blowup views

The blowup views contain everything including random and unplaced chromosomes.

Create blowup output directories.

```
mkdir -p blowups
cd blowups
```

Generate a gap annotation file.

```
fasta_findNs.pl -i ../fastawhole -o gap.log
sed -i 's/.fa//' gap.log
```

Filter by length of N blocks. Here gaps need to be >5Kb. Change "5000" to the number of basepairs you want to filter by.

```
head -n 1 gap.log > 5kgap.extra
awk '$3 - $2 > 5000' gap.log >> 5kgap.extra
```

Test the blowup configuration with a single chromosome (e.g., chr1).

```
/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull \
    5kgap.extra 1000 0.90 chr1 1 ../showseq.out
```

Use /home/linchen2/ps/wgac/pantro2/batchblowUp.pl to generate a script file blowupView. However, you have to modify the file to generate the appropriate command which calls appropriate perl program chromosome_blowups_####.pl

Use `/home/linchen2/ps/para/autoGenPs.pl` to read the file and execute scripts to generate blowup images.

Generate the blowup images at 90, 94, 98, and 99% identity thresholds and 5 Kbp length threshold.

```
/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 5000 0.90 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 5000 0.94 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 5000 0.98 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 5000 0.99 all 1 ../showseq.out
```

Repeat with a 10 Kbp length threshold.

```
/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 10000 0.90 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 10000 0.94 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 10000 0.98 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 10000 0.99 all 1 ../showseq.out
```

Repeat with a 20 Kbp length threshold.

```
/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 20000 0.90 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 20000 0.94 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 20000 0.98 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 20000 0.99 all 1 ../showseq.out
```

Repeat with a 40 Kbp threshold.

```
/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 40000 0.90 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 40000 0.94 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 40000 0.98 all 1 ../showseq.out

/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 40000 0.99 all 1 ../showseq.out
```

Generate a blowup view for duplications at least 100 Kbp long and with an identity of at least 95%.

```
/net/eichler/vol4/home/jlhudd/wgac/chromosome_blowups.pl ../data/oo.weild10kb.join.all.cull 5kgap.extra 100000 0.95 all 1 ../showseq.out
```

Convert PostScript files to PDF while maintaining the same folder structure. Pass the parent of the starburst folder as an argument.

```
perl /net/eichler/vol4/home/jlhudd/wgac/dirBlowupP2p.pl `pwd`
```

Visualize WGAC with the UCSC browser

The WGAC table, `GenomicSuperDup.tab`, can be loaded into the UCSC database in a local table. This approach relies on code in the UCSC browser that looks for a table named exactly "GenomicSuperDups" and knows how to display that type of information.

This approach does not work for track hubs and custom tracks. For these methods, you can simulate the appearance of the built-in GenomicSuperDups track using the BED format.

Use the following command to convert `GenomicSuperDups.tab` to a BED file where the name column has the name of the other duplication copy, the score has the % identity of the alignment multiplied by 1000 (to fit the expected score range of 0-1000), and the strand has whether the duplications are in direct (+) or reverse (-) orientation with respect to each other.

```
awk 'OFS="\t" {
    if ($6 == "_") { strand="-" }
    else { strand="+" }

    if ($26 > 0.99) { color="255,140,0" }
    else if ($26 > 0.98 && $26 <= 0.99) { color="255,255,0" }
    else { color="105,105,105" }

    score=sprintf("%i", $26 * 1000);
    print $1,$2,$3,$4,score,strand,$2,$3,color
}' GenomicSuperDup.tab | sort -k 1,1 -k 2,2n > segmental_duplications.bed
```

To view these data in a track hub, convert the BED file to a bigBed file.

```
module load ucsc/20140617
bedToBigBed segmental_duplications.bed contigs.tab segmental_duplications.bb
```

You can view this file in an existing track hub like so:

```
track wgac
shortLabel Segmental duplications
longLabel Segmental duplications
type bigBed 9
bigDataUrl segmental_duplications.bb
itemRgb on
```

---

methods/pipelines/wgac.txt · Last modified: 2022/04/27 17:36 by David Gordon