# RedDog v1.0b - Instruction Manual

Authors: D. J. Edwards, B. J. Pope and K. E. Holt
Date: March 25, 2015

## Copyright Notice



## Citing RedDog:

Edwards, D. J., Pope, B. J., and Holt, K. E. (2015) RedDog: comparative analysis pipeline for large numbers of bacterial isolates using high-throughput sequences. *In prep.*

## Table of Contents

## Obtaining the Current Version of RedDog

The latest version of the pipeline is available from https://github.com/katholt/RedDog

Make sure that you obtain the 'Master' version, which is always the latest, tested release version of the pipeline, and not one of the development branches.

**To download the latest version from Github:**
*Note: these instructions assume you are running the pipeline on a Unix-based cluster environment. Currently, there are versions of Rubra that support SLURM and Torque systems. RedDog is being developed on a SLURM cluster at VLSCI.*

Change directory (`cd`) to your directory on the drive where you will keep the new version of the pipeline. Then:

```
module load git
git clone https://github.com/katholt/RedDog RedDog_v1b
```

You will be asked for your Github account name and password. This command will then create a new folder named `'RedDog_v1b'` with the latest version of the pipeline.

## Reporting a Problem with RedDog

**If you want to report a problem with RedDog, or to suggest a change or improvement, please use the RedDog issues page (https://github.com/katholt/RedDog/issues) on Github.**

When reporting a bug, be sure to include any relevant error messages, the version of the pipeline being used, and the location of the pipeline and the log files (if not within the pipeline folder).

**Note:** Errors during a run produced by a job failing to finish due to walltime or memory limits should *generally not be reported*.

First, try extending the walltime or memory (which ever is applicable) of the stage in question (also take into account other stages that may be affected – see below in "Large Data Sets, Large Read Sets and High Variation") and restart the pipeline – in most cases the pipeline will then take up where it last left off, redoing the failed jobs with the now extended walltime/memory. If on trying this, the pipe fails, then report the problem. See 'Troubleshooting' for more information on what to do if things do go wrong.

## RedDog Dependencies

| | |
|---|---|
| Python v2.7.5 | https://www.python.org/ |
| Ruffus v2+ | http://www.ruffus.org.uk/ |
| Rubra | https://github.com/bjpop/rubra |
| Biopython v1.6+ | http://biopython.org/wiki/Main_Page |
| Bowtie2 v2.2.3 | http://bowtie-bio.sourceforge.net/bowtie2/index.shtml |
| BWA v0.6.2 | http://bio-bwa.sourceforge.net/ |
| SAMTools v1.1 | http://samtools.sourceforge.net/ |
| BAMTools v1.1 | https://github.com/pezmaster31/bamtools |
| eautils v1.1.2 | https://code.google.com/p/ea-utils/ |
| FastTree v2.1.7** | http://www.microbesonline.org/fasttree/ |

The list above does not include any further dependencies required by any of the above packages to run correctly – make sure you carefully follow the instructions for each when installing. **Make sure to install the double-precision version of FastTree (See the FastTree FAQ and this blog-post for more details).

## Using RedDog on a Dedicated Server

If you are using RedDog on a non-distributed system, i.e. a dedicated server with no queuing system, there is one change needed before running RedDog, and one that should be considered.

First, the needed change; in the config file, look for:

```
stageDefaults = {
    "distributed": True,
```

Change the highlighted setting to:

```
"distributed": False,
```

The second is the number of jobs to be launched ; in the config file, look for:

```
pipeline = {
    "logDir": "log",
    "logFile": "pipeline.log",
    "style": "print",
    "procs": 50,
```

If your server has less than 50 processors available, you should change this to a suitable number. You may also want to keep in mind that certain stages can require up to 4 Gb for processing.

## Running the Pipeline

**WARNING 1: IT IS HIGHLY RECOMMENED THAT ONLY ONE RUN OF THE PIPELINE BE RUN AT A TIME - ESPECIALLY WITH LARGE DATA SETS.**

**WARNING 2**: The pipeline analysis for larger data sets can take **many, many hours**, so set it up on a computer you can leave running uninterrupted (you can still do other tasks on the computer, though I often turn off the screen and come back later), or make use of the screen command (see below)

Change to the RedDog directory, then at the command prompt, enter:

```
module load python-gcc/2.7.5
```

You only need to load the module once per login session. Once the module is loaded (almost immediately), enter:

```
rubra RedDog --config RedDog_config
```

This will print out all the stages as they will be run and is useful for confirming the details of a run before commencing it. To actually run the pipeline, enter:

```
rubra RedDog --config RedDog_config --style run
```

This will start the pipeline, and after a series of pre-run checks, the details of the run will be reported and the user asked to start the run. Once you have checked the details, hit 'y' to start the actual pipeline. The pipe will then launch a series of job scripts that will be sent to the job queue for processing.
*e.g*. Example run

```
RedDog V1.0beta - phylogeny run

Copyright (c) 2015, David Edwards, Bernie Pope, Kat Holt
All rights reserved. (see README.txt for more details)

Mapping: Bowtie2 V2.2.3
Preset Option: --sensitive-local
1 replicon(s) in GenBank reference NC_000962_3
1 replicon(s) to be reported
768 sequence pair(s) to be mapped

Output folder:
/scratch/disk/workspace/mapping/v1b_date_study/

Start Pipeline? (y/n)
```

If your connection to the cluster system is broken during the run, don't panic. Just log back into a new session and rerun the above two commands again. The pipeline should restart at the stage the pipe was at before the connection was broken. *(If it doesn't, report the problem)*

If you want a flowchart (only svg files are currently available - open with any browser):

```
rubra RedDog --config RedDog_config --style flowchart
```

You can also run the pipeline using the 'screen' command at the command prompt. This opens a new screen connection in the same terminal session, and you will get the 'welcome' message. You will be in the same directory as when you entered `'screen'`. VLSCI users: If you do want to run the pipeline in the screen session, you will have to do the two following steps when you start a new screen session:

```
module purge
module load vlsci
```

Then change to the RedDog directory (using '`cd`'), load the python module as usual, and then launch the pipeline.

You can detach from the screen by entering `crtl-a d` (hit 'a' while holding the 'control' key, then hit 'd' by itself). This will return you to the main session you logged on with. You can exit this (and turn off your computer and go home) and the pipeline will continue to run in the 'screen' session.

To re-attach to the session, just type in `screen -r` to recover the session. You can do this from any computer, but do remember to detach if you are just checking from home, for example. Screen sessions will keep running even if your connection to your host is broken for some reason. For more details on screen, including running more than one screen session and naming sessions, look at the following:
Screen quick reference http://aperiodic.net/screen/quick_reference

There is one important caveat to using screen; it does not hold a record of the lines as it goes so you can't scroll back to examine an error for example. Using `crtl-a H` (hit 'a' while holding the 'control' key, then hit 'H' by itself) creates a running log of the session. Screen will keep appending data to the file through multiple sessions. Using the log function is very useful for capturing what happens during the run. If something goes awry, you can look back through your logs.

For a further, user-friendly guide to screen (and source of the final 'hint') visit:
How To Use Linux Screen: http://www.rackaid.com/blog/linux-screen-tutorial-and-how-to/

## Pipeline Conventions

Before using RedDog, and reading this manual, it is useful to understand some of the conventions used in this pipeline.

The pipeline can map reads to a reference with more than one sequence, (hereafter these sequences are referred to as **replicons**). The replicons in a reference can be made up of a whole bacterial 'chromosome' with (or without) one or more plasmids, or a series of plasmids, bacteriophage or genes of interest, or indeed a series of contigs (*e.g*. a pangenome sequence). If this reference is given in Genbank format, the pipeline will also examine the depth and coverage of reads for each isolate across all the features (genes) in the Genbank file.

Where the replicons in the reference represent sequences with potentially different phylogenetic histories (*e.g.* a genome and one or more plasmids), the pipeline will report on each replicon in the reference separately, including calling potential outgroups – this is known as a '**phylogeny**' run.

If the replicon(s) represents the pangenome of the collection of isolates to be tested, then only the replicon(s) with the 'core' genes (and SNPs) needs to be considered in phylogenetic analysis. This is usually the largest contig, but the user can specify any one replicon, or more. During such a '**pangenome**' run, all isolates are assumed to have been used to generate the pangenome, and hence are all isolates are classified as 'ingroup' (*i.e.* there is no outgroup calling).

Mapping runs to a new reference are called a **'new' run**; this is to distinguish them from a run where any new read sets are merged with the output from a prior run, a so called **'merge' run**. As long as the same reference is being used, any number of merge runs can be added to the original output set. Merge runs can also include mapping reads from different platforms/technologies.
*e.g.* run 1: 'new' with paired end Illumina data;
run 2: 'merge' with single end Illumina data;
run 3: 'merge' with more paired end reads; and,
run 4: 'merge' with Ion Torrent reads.

## Pipeline Test Sets

The following small set of reads and reference Genbank file are recommended to use to test if the installation is working properly. The read sets are available for download at the European Nucleotide Archive (http://www.ebi.ac.uk/ena).

Read Sets: ERR019786, ERR019793 and ERR019794

The reference genome and plasmid are available from NCBI.

Reference: CP000038 and CP000039
(http://www.ncbi.nlm.nih.gov/nuccore/CP000038.1,CP000039.1)

## The Configuration File

At the heart of any RedDog pipeline run is the **RedDog_config** file. This file contains all the inputs, options and commands for the pipeline.

The four most important inputs are the reference file, sequences to be mapped, the output folder for the run, and, if a merge run is being executed, the name of the folder with the output from the prior run. During a merge run, the results of mapping any new reads will eventually be merged into this prior output folder, updating the results with the expanded data set.

```
'''
Configuration file for RedDog.py V1.0beta
-----------------------------
Essential pipeline variables.
'''
reference = "/full/path/to/folder/ref_with_plasmid.gbk"

sequences = "/full/path/to/reads/fastq_folder/*.fastq.gz"

output = "/full/path/to/folder/RedDog_output/<ref>_<date>/"

out_merge_target = ""
#out_merge_target = "/full/path/to/folder/RedDog_output/<ref>_<date>"
```

### Essential pipeline variables

#### "reference"

Can be Genbank or fasta format, and contain one or more replicons - if Genbank format, this will be converted to a fasta version for mapping. If the Genbank file is not given, the gene cover and depth matrices for genes will not be generated and nor will SNP consequences. If you don't have the Genbank record, or don't want the above matrices to be generated, use a fasta format reference instead. You don't need to tell the pipeline which type of reference you have – it will work it out.

Note: if you download a fasta file from Genbank, you will need to simplify the header for each replicon in the reference before running the pipeline. Also, each replicon is required to have a unique name.

### "sequences"

Sequence reads to be mapped; you can point to the folder and the set of files to be mapped with a wildcard qualifier:

```
sequences =  '/full/path/to/reads/*.fastq.gz'
```

You can also combine sequences from different folders into the same run:

```
sequences = ['/full/path/to/reads/set1/*.fastq.gz',
             '/full/path/to/reads/set2/*.fastq.gz']
```

Note that the pipeline expects the reads to be in `fastq` format and stored as `gzip` files. The pipeline can take reads from Illumina platforms, as either single end or paired end reads, or Ion Torrent sequences, though Illumina paired end sequences are the default setting (see 'readType' below for more details). Each read set is required to have a unique name.

### "output"

Full path name for the output folder. Note that this folder should not exist at the start of any run (new or merge).

### "out_merge_target"

When running a 'new' analysis, set to null string. Otherwise set to the directory you want to merge with. You can only merge a prior run with a new run (not two prior runs).

This 'merge target' folder must have the bams and indexes in one sub-folder (`./bam`) and the vcfs in another (`./vcf`). There also must be a '`sequence_list.txt`' file, and preferably (but not necessarily) a '`<reference>_run_report.txt`' file - if a previous run_report.txt is available, this information will be used to ensure continuity in settings between merge runs. (See **Outputs** for more information)

The 'output' folder (see above) for a merge run should NOT exist prior to the run, and will be deleted at completion of the pipeline.

## Other pipeline variables

If none of the following variables are set or changed, the default settings will be used.

### Default Settings

The default settings are:

Illumina paired-end sequences (PE)
runType by replicon number
SNP calling for largest replicon (pangenome)
Bowtie2 'sensitive-local' mapping
Bowtie -X option = 2000
minimum read depth = 5
cover_fail = 50%
depth_fail = 10
mapped_fail = 50%
sd_out = 2
check_reads_mapped for largest replicon (phylogeny)
95% conservation for SNP table
hetsVCF = False (off)
DifferenceMatrix = False (off)

Each of these pipeline variables is explained in more detail below.

### "readType"

Choose the type of input sequences:

IT for Ion Torrent (single reads),
SE for Illumina single-end reads, or
PE for Illumina paired-end reads

Each read type has a particular pattern you need to follow for use in the pipeline:

IT: *_in.iontor.fastq.gz
SE: *.fastq.gz
PE: *_1.fastq.gz and *_2.fastq.gz

(*i.e.* forward and reverse reads in separate files for PE read sets)

```
readType = "PE"
Or:
readType = "SE"
Or:
readType = "IT"
```

### "runType"

If set to a null string (*i.e.* `runType = ""`) then the number of replicons in the reference will determine the run type:

<= 100 replicons (*e.g.* reference genome + plasmids + phage) - phylogeny run type
> 100 replicons (*e.g.* multifasta pangenome) - pangenome run type

The user can choose to override the run type, by setting it as shown here:

```
runType = "pangenome"
```

Or:

```
runType = "phylogeny"
```

This difference between these two runTypes is described above in "Pipeline Conventions".

### "core_replicon"

For a pangenome run, the SNPs will only be called for the largest replicon - this assumes the core genome is to be found in this replicon. The user can define an alternative replicon (or replicons) for the SNP calling.

Notes: there must be a space after any comma, and set to null string to get the largest contig

*e.g.*

```
core_replicon = "NC_007384, NC_007385"
```

Or:

```
core_replicon = ""
```

### "mapping"

For Illumina reads, both BWA sampe/samse and Bowtie2 are available. If you don't set the mapping, the default is Bowtie2.

| | Illumina | | |
|---|---|---|---|
| | PE | SE | IT |
| BWA sampe | Y | N | N |
| BWA samse | N | Y | N |
| Bowtie2 | Y | Y | Y |

Note: You cannot use two mappers during the same run!

*e.g.*

```
mapping = "bwa"
```

Or:
`mapping = “bowtie”`

### “bowtie_map_type”

When using Bowtie2 mapping, it is possible to change the preset mapping options used with the 'bowtie_map_type' option.

Preset options in --end-to-end mode:
--very-fast
--fast
--sensitive
--very-sensitive

Preset options in --local mode:
--very-fast-local
--fast-local
--sensitive-local
--very-sensitive-local

Default: if bowtie_map_type is set to “” sensitive-local mode mapping will be used.

See the Bowtie2 Manual for more information on the presets:
http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml

### “bowtie_X_value”

When using Bowtie2 mapping, it is also possible to change the -X option in Bowtie2 using with the 'bowtie_X_value' option. The default maximum length of -X for bowtie2 to consider pair-ended reads contiguous is 2000.

*e.g.*
`bowtie_X_value = 1500` (this is the same value as used in BWA)

### “SNPcaller”

With the advent of BCFtools v1+, SNP calling can now be accomplished using two different algorithms; consensus caller ["c"] (original) or multiallelic caller ["m"] (new bcftools v1+)

Note: multiallelic caller is used only for calling unique SNPs from the BAM files.
The consensus sequences used to populate the allele table based on these SNPs are still generated using the original consensus caller - this will be changed if/when a vcf2fq program is available for multiallelic-generated VCFs.

Unless you have a specific reason to use the multiallelic caller, it is strongly recommended to stick with consensus SNP calling for now.

*i.e.*
`SNPcaller = "c"`
Or:
`SNPcaller = "m"`

### "replaceReads"

You can also "remove" one or more read sets: these will be marked as "failed" rather than actually removed. This option is only available during a merge run.

*e.g.* replace a set of reads with their "qc-ed" version
`replaceReads ="read_set_2,read_set_3,read_set_24"`

Note: there is no space between reads, just a comma.

### "minimum_depth"

Minimum depth of reads for variant calling. (SAMtools)

Default value: minimum_depth = 5

### "HetsVCF"

The pipeline filters out heterozygous SNP calls. To capture these SNPs in the form of a VCF (one per isolate), set HetsVCF to 'True'.

e.g.
`HetsVCF = False` (default: no VCF produced)
Or:
`HetsVCF = True` (VCF produced)

### "cover_fail"

If the percentage of the reference replicon mapped by reads from a particular isolate is below this threshold, the isolate will be called a 'fail' and removed from downstream SNP analysis.

Default value: cover_fail = 50%

### "depth_fail"

If the average read count for any reference replicon falls below this threshold value, the isolate will be called a 'fail' and remove from downstream SNP analysis.
Default value: depth_fail = 10

### "mapped_fail"

If the percentage of total reads mapped to the replicon falls below this threshold value, the isolate will be called a 'fail' and removed from downstream SNP analysis.
Default value: mapped_fail = 50%

### "sd_out"

For any isolate, if the SNP count is more than sd_out*s.d. from the mean count, the isolate will be called as an outgroup (phylogeny run only).
Default value: sd_out = 2

So for the default value, any isolate with greater than the mean SNP count + 2*s.d. will be called an outgroup.

### "check_reads_mapped"

To switch on or off the checking of percentage of reads mapped, use the following 'check_reads_mapped'.

By default (*i.e.* set to the null string, "") the pipeline will use the largest replicon.
If it is set to "off", there will be no check for percentage of reads mapped. Otherwise, give list of the n replicons to be checked, followed by an 'x', then followed by the ratio of the first n-1 replicons. For a single replicon, just put the replicon.

*e.g.*

```
check_reads_mapped = ""
```

Or:

```
check_reads_mapped = "off"
```

Or:

```
check_reads_mapped = "rep_1"
```

Or:

```
check_reads_mapped = "rep_1,rep_2,rep_3,x,0.45,0.3"
```

*i.e.* In the last example, rep1 is 45% of the total genome, rep2 is 30% of the total genome, and rep3 is 25% of the total genome (by default). Note: there must be no spaces in the list.

### "conservation"

During allele matrix filtering, you can set the conservation level for missing alleles. This is a ratio between 1.0 (100% conservation - remove all SNPs with even one missing allele call) and 0.0 (0% conservation - remove no SNPs). By default, the pipeline produces the 95% and 0% conservation matrices, with downstream analysis on the 95% matrix.

By entering a different conservation level (*e.g.* 0.85), both the 95% and 0% matrices will still be produced, but so too will the 85% matrix (in this example), and downstream analysis carried out on this latter matrix.
Default value: conservation = 0.95

### "DifferenceMatrix"

The pipeline can produce a difference matrix as optional output. Currently this is a pairwise difference count. To get the difference matrix, set 'DifferenceMatrix' to 'True'.

*e.g.*
`DifferenceMatrix = False` (default: No difference matrix produced)
Or:
`DifferenceMatrix = True` (difference matrix output produced)

## Advanced Settings

In this part, we will continue to look at the configuration file. However, if you have a small number of samples, you can probably skip this.

**If you have a large numbers of reads for analysis (over 150), any read files are large (over 800Mb), or a large number of SNPs are expected (high variation within study group – 10,000s of SNPs), take note of the necessary changes following below.** Sometimes, some read sets will behave like larger read sets – why has still not been pinpointed. However, if a smaller read set fails the mapping due to walltime, it's probably a good idea to treat this read set like a large set.

## Rubra pipeline variables

+ logDir: the directory where batch queue scripts, stdout and sterr dumps are stored.

+ logFile: the file used to log all jobs that are run.

+ style: the default style, one of 'flowchart', 'print', 'run', 'touchfiles'. Can be overridden by specifying --style on the command line.

+ procs: the number of python processes to run simultaneously. This determines the maximum parallelism of the pipeline. For distributed jobs it also constrains the maximum total jobs submitted to the queue at any one time.

+ verbosity: one of 0 (quiet), 1 (normal), 2 (chatty). Can be overridden by specifying --verbose on the command line.

+ end: the desired tasks to be run. Rubra will also run all tasks which are dependencies of these tasks. Can be overridden by specifying --end on the command line.

+ force: tasks which will be forced to run, regardless of timestamps. Can be overridden by supplying --force on the command line.

+ rebuild: one of 'fromstart','fromend'. Whether to calculate which dependencies will be rerun by working back from an end task to the latest up-to-date task, or forward from the earliest out-of-date task. 'fromstart' is the most conservative and commonly used as it brings all intermediate tasks up to date.

```
pipeline = {
    "logDir": "log",
    "logFile": "pipeline.log",
```

```
    "style": "print",
    "procs": 100,
    "paired": True,
    "verbose": 1,
    "end": ["deleteDir"],
    "force": [],
    "rebuild" : "fromstart"
}
```

In the above example (the default settings), the pipeline will run from start to completion (as long as there are no errors). This is signified by the '"end": ["deleteDir"]' switch. You can use this switch to pause the pipeline at a particular stage by changing it to the stage you want to halt at.

'style' - allows for the type of output from the pipeline to change - usually you will want 'run', where the pipeline does the analysis, but you can also change to 'flowchart' or 'print'. An example flowchart for a new run using Illumina paired-end sequences can be found at the end of this document. The 'print' style will print out a list of the tasks to be undertaken in the run.

### Large Data Sets, Large Read Sets and High Variation

A number of factors can impact on the performance of the pipeline, causing it to halt when jobs run past the allocated walltime or memory allocation.

Fortunately, the pipeline is designed to be able to recover from this situation. (See also 'troubleshooting' below.) First, try extending the walltime or memory (which ever is applicable) of the stage in question (also take into account other stages that may be affected, and restart the pipeline – in most cases the pipeline will then take up where it last left off, redoing the failed jobs with the now extended walltime/memory.

To provide for users with larger data sets *etc.*, there is a second configuration file provided with all the settings with more relaxed specifications – 'large_RedDog_config.py''. Whilst this may result in a slower pipeline than the other configuration file (especially in a busy job queue), each job is much less likely to fail on walltime/memory allocation, so the pipe more likely to finish without user intervention. Otherwise, the user can use the instructions that follow to 'tune' the pipeline for their situation.

How the pipeline is affected by these factors is summarised in a table below.

### *Large Data Sets*

Scrolling down towards the end of the configuration file, you will find the following stages that need changing when analysing larger data sets (more than 150 samples). The stages of the pipeline specifically affected by the number of isolates being mapped include:

| | |
|---|---|
| parseSNPsNoGBK | walltime |
| getDifferenceMatrix | walltime (can need more memory for very large sets) |
| makeTree | walltime and memory |

The following stages are also affected if a Genbank reference is used for mapping:

| | |
|---|---|
| parseSNPs | walltime |
| collateAllRepGeneCover | walltime |
| mergeAllRepGeneCover | walltime (only in merge run) |
| parseGeneContent | walltime (can need more memory for very large sets) |

For instance, “collateAllRepGeneCover” only needs a walltime of 10 minutes for small data sets, but for larger sets needs more walltime to complete. Similarly, “makeTree” (using FastTree) takes less than 15 minutes, but for a larger data set can take up to 12 hours and require much more than the default 4Gb of memory. For large data sets, make sure you change the 'walltime', and the 'memInGB' where indicated, to the suggested values before you start the pipeline. For really large data sets, you have to extend these even further.

*e.g.* In the stage ‘getDifferenceMatrix’:

```
    "getDifferenceMatrix": {
        "walltime": "00:10:00",
# large data sets
#         "walltime": "06:00:00",
        "command": "python make_distance_matrix.py %in"
    },
```

Change the highlighted part to:

```
#         "walltime": "00:10:00",
# large data sets
        "walltime": "06:00:00",
```

### *Large Read Sets*

Size of individual read sets can also impact on performance of some stages, again causing it to halt when jobs run past the suggested walltime or memory.

The “threshold” size for this impact is about >850Mb (individual read sets – not the paired set). However, as already noted above, sometimes small read sets will behave like larger read sets – we still have not pinpointed why this is the case. However, if a smaller read set fails the mapping (the first stage below) due to walltime, it’s probably a good idea to treat the read set like a large set. Pipeline stages affected by this include:

alignBowtiePE/alignBowtie/alignBWAPE/alignBWASE (as appropriate)
callRepSNPs
getCoverage
getConsensus
getMergeConsensus (only during merge run)

### *High Variation*

The number of SNPs found within bacterial species can vary greatly, from very high variance in sequence found in, for example, *Acinetobacter* and *Klebsiella* species to the much lower variance in, for example, TB species. Those species with high divergence between isolates will have much higher SNP counts, which can in turn impact on walltimes and memory needs of certain stages of the pipeline. Pipeline stages affected by this include:

| | |
|---|---|
| deriveRepAlleleMatrix | walltime |
| parseSNPsNoGBK/parseSNPs | walltime |
| getDifferenceMatrix | walltime (and memory if large set as well) |
| makeTree | walltime and memory |

If you have a large number of read sets to map, that include some sets of reads that are themselves really large, and the bacteria being mapped is also highly variable, then those stages that appear in more than one list may need to be boosted even further for successful running.

## Summary Table - Large Data Sets, Large Read Sets and High Variation

| Stage | Large Data Set (>150 isolates) | Large Read Sets (>850Mb *[1]) | High Variation |
|---|---|---|---|
| **alignBowtiePE/ alignBowtie/ alignBWAPE/ alignBWASE** *[2] | | W | |
| **callRepSNPs** | | W | |
| **getCoverage** | | W | |
| **getConsensus** | | W | |
| **getMergeConsensus** | | W* | |
| **deriveRepAlleleMatrix** | | | W |
| **parseSNPsNoGBK/ parseSNPs** *[3] | W | | W |
| **getDifferenceMatrix** | W** | | W*** |
| **makeTree** | WM | | WM |
| *collateAllRepGeneCover* | W | | |
| *mergeAllRepGeneCover* | W* | | |
| *parseGeneContent* | W** | | |

Legend and Notes:
W - extend walltime
W* - extend walltime, merge run only
W** - extend walltime, memory as well for very large sets
W*** - extend walltime, memory as well for large sets
WM - extend walltime and memory

*[1] Sometimes a smaller read set can behave like a large read set
*[2] alignBowtiePE or alignBWAPE for PE Illumina read sets,
alignBowtie or alignBWASE for SE Illumina read sets,
and alignBowtie for Ion Torrent
*[3] parseSNPsNoGBK for FASTA reference, parseSNPs for Genbank reference

Last three stages (in *italics* above) for Genbank reference only

## Clean-up

There are a large number of files, both large and small, that are produced during the running of the pipeline, and most should be deleted straight after a successful run to save disk space (and keep your account under control – trust me). Fortunately, the pipeline will do most of this for you - at least the 'temp' directory.

You do still need to do some housekeeping between runs of the pipeline. The log folder for the run (typically created in the RedDog directory) should be deleted in entirety. **DO NOT DELETE THIS LOG FOLDER WHILE THE PIPE IS RUNNING!**

At the end of each run, there is one file, "finish.deleteDir.Success", that does need to be removed - especially so if the run will be merged with other read sets later. Since version 0.5, the pipe will not launch a merge run unless this file has already been removed from the 'out_merge_target' folder.

Note: In the RedDog directory after the run, you can ignore the *.pyc files. They are small, and recreated at the start of each run, so deleting them is a futile exercise. (Though most times I can't help myself...)

## Pipeline Outputs

If a reference FASTA file is provided in the config file, the following outputs are produced:

| | | |
|---|---|---|
| BAM files and indexes | 'bam' subfolder | <isolate>.bam, <isolate>.bam.bai |
| VCF files* | 'vcf' subfolder | <isolate>_q30.vcf |
| **Run statistics - summary** | Output folder | <reference>_AllStats.txt |
| **Run statistics by replicon** | Output folder | <reference>_<replicon>_RepStats.txt |
| SNP allele matrix (0% cons) | Output folder | <reference>_<replicon>_alleles_var.csv |
| **SNP allele matrix (95% cons)** | Output folder | <reference>_<replicon>_alleles_var_cons0.95.csv |
| SNP difference matrix ** | Output folder | <reference>_<replicon>_SNP_diff.nxs (NEXUS format) |
| Concatenated SNPs | Output folder | <reference>_<replicon>_alleles.mfasta |
| Tree (concatenated SNPs) | Output folder | <reference>_<replicon>_alleles.tree (NEWICK format) |
| Outgroups | Output Folder | <reference>_<replicon>_outgroup.txt *** |
| Consensus Warning | Output Folder | <replicon>_<isolate>_cns_warning.txt **** |
| Run Report | Output Folder | <reference>_run_report.txt |
| Sequence List | Output Folder | sequence_list.txt |

If a reference Genbank file is provided instead of a FASTA reference in the config file, the following outputs are also produced:

| | | |
|---|---|---|
| **Percentage cover by gene** | Output folder | <reference>_CoverMatrix.csv |
| Depth by gene | Output folder | <reference>_DepthMatrix.csv |
| Gene Presence/Absence | Output folder | <reference>_PresenceAbsence.csv |
| **SNP consequences outputs** | Output folder | <reference>_<replicon>_alleles_var_cons0.95_consequences.txt, <reference>_<replicon>_alleles_var_cons0.95.gbk |

Notes:
* optional output includes <isolate>_hets.vcf if 'HetsVCF' set to 'True'
** optional output only generated if 'DifferenceMatrix' set to 'True'
*** outgroup.txt will only be generated if there are any outgroups called for a replicon set.
**** cns_warning.txt only generated if SAMTools fails to generate a consensus sequence for an isolate (happens on the very odd occasion with smaller replicons, *e.g.* plasmids).

If the allele conservation is set to something other than 0.95, this will be reflected in the file names (where applicable)

For further information on those output files highlighted in bold above, see **Understanding the Output - Examples**.

## Further Analysis: parseSNPTable.py and SNP allele tables

Once the pipe has finished running, probably the most important output is the SNP allele matrix. There are two versions of the table produced for each replicon in a phylogeny run, or the largest/user-specified replicon(s) in a pangenome run. One is the table with no conservation of alleles applied, and the other the 95% conservation of alleles table (Note: if the user chooses to use a different conservation level [*e.g.* 99%], a third table of alleles with the appropriate level of conservation across alleles will be produced too.

Whilst one can use the 95% conservation allele to produce a 'core' phylogeny, there are a number of factors that need to be considered and dealt with. Some regions of the sequence need to be excluded as they are repeated within the genome, or region(s) of recombination. Or the researcher may want to exclude any phage. Or, indeed, any combination of the three. Or only focus on a particular region of the genome. Also, whilst the pipeline does call outgroups, this call is not applied to the allele table (and hence nor the resulting tree). Calling of the outgroup(s) is left for the user to apply post-run.

To produce an allele table (and sequence alignment) dealing with these factors, the user can apply the parseSNPTable.py script found in the RedDog pipeline folder.

To get help with the commands for parseSNPTable.py, use:

python parseSNPtable.py -h

Required inputs:
-s SNP alleles (in csv format)
-p Prefix for output files
-m Modules to run (see below)
Optional input:
-d directory for output files (if different from current working directory)

Defining strains and outgroups to use:
When the SNP table is read in, it will be screened to include only the strains that you want to know about, and only SNPs that are variable within this group.
-l file containing a list of strains to include (otherwise all strains in the input file are included in the outputs)
-o outgroup(s), comma separated list. If specified, SNPs that are non-variable in the ingroup are removed first, and variation in an outgroup(s) is ignored in assessing conservation. Note it doesn't matter whether or not an outgroup(s) is included in the list of strains.
-g Gap character that designates unknown/missing allele in the table. Default is '-' (output by our pipelines), some other pipelines use 'N'.

The table is then parsed as specified by the modules in -m, options include:
aln - convert to fasta alignment

filter - filter SNPs that are included/excluded in regions specified via -x (genbank, gff or 2-column CSV table format)

clean - filter out any pairs of SNPs with -P bp between them (default 3bp, minimum 2bp) and any trio or more of SNPs within -W bp in any isolate (default 10bp, minimum 3bp or -P if greater than 3)

cons - filter SNP positions that are not conserved above a cutoff specified via -c (e.g. -c 0.99 -> all snps with >1% missing alleles is filtered out)

core - filter SNPs that are not in the core genes. Core genes are those found in the core isolates (list -L) that have coverage of greater than 90% (option -Z, % coverage as a ratio) found in the gene coverage matrix (-z)

vcf - convert snp table to vcf format (requires -v name of reference mapped, optional -V name of isolate mapped, otherwise -v used). Can also produce VCF with SNPs filtered out at each stage (set '-A True' and add vcf after each filtering step)

coding - annotate SNPs with their coding effects (must specify reference genbank file that contains the sequence and CDS features, via -r : use -q to specify particular reference sequence to apply from multiple reference genbank files)

Specify modules in a comma-separated list, *e.g.* '-m filter,cons,aln'

Any number of these modules can be supplied in any order; the order they are given is the order they will be run; *e.g.* '-m filter,cons,aln' will read run region filtering, then conservation filter, then make a fasta alignment.

aln: Alignment
The current SNP allele table is printed to an alignment in fasta format

filter: Specify regions of SNPs to include or exclude from outputs
-x file specifying regions of the genome; these coordinates will be compared against those of the SNP allele table and used to determine whether each SNP should be included in the outputs or excluded from the outputs.
-y include

So, if you do -x core_genes.gbk -y include, then all SNPs within these regions (core genes) will be included in the output tables & alignments, and all SNPs outside these regions (ie non core sequences) will be excluded.

Alternatively, if you do -x phage_and_repeats.gbk -y exclude, then SNPs in these regions (phage sequences and repeats) will be excluded from the output files.

Possible formats for the regions file (-x), recognised by their extension, are:

genbank (.gbk or .gb)

All features of any type (misc_feature, CDS, etc) will be read in as regions to include/exclude.

GFF (.gff)

Assumes file is tab-delimited and the start and stop coordinates are in columns 4 & 5.

text (.txt)

Assumes file is tab-delimited and the start & stop coordinates are in columns 3 & 4 (*i.e.* format output by the genbank2table.py script)

csv (any other extension)

Assumes file is comma separated and has start and stop coordinates in columns 1 & 2

clean: filter out any pairs of SNPs with -P bp between them (default 3bp, minimum 2bp) and any trio or more of SNPs within -W bp in any isolate (default 10bp, minimum 3bp or -P if greater than 3)

-P size of pairs window (int, default 3)

-W size of larger window (int, default 10)

cons: Filter SNP loci that are not present above a specified proportion

-c Minimum proportion of allele calls required to retain SNP (default 0.99, *i.e.* all SNPs for which >1% of strains have unknown alleles are removed)

vcf: converts snp table to vcf format

-v name of the reference mapped (default none, use MT for PLINK)

-V name of the isolate mapped (default -v)

-A set to get VCF with PASS/FAIL for all SNPs (default False)

core and coding: Specifying a reference (required for either)

-r genbank Note this must contain both the sequence, and the annotation of features

-q sequence name - Sequence in multiple reference Genbank file that applies to SNP table

core: filter SNPs that are not in the core genes.

-L file containing a list of strains to include in core (otherwise all strains in the input file are included sans outgroups)

-z gene coverage file (RedDog output: csv)
-Z minimum % coverage of each gene (as ratio) across core strains required to retain SNP locus (default 0.9)

coding**:** Additional options.
-f Feature types that designate protein coding sequences (will determine whether each SNP in these regions is synonymous or non-synonymous); default CDS
-e Feature types to exclude from SNP annotation; default gene,misc_feature
-i Unique identifier for features, this is the feature name used in the output file. Must be unique; usually 'locus_tag' is appropriate.

By default, we assume that translatable gene sequences have the feature type 'CDS' and we don't want to annotate whether SNPs are in 'misc_feature' features (as these are probably irrelevant) or 'gene' features (as these are often duplicates of 'CDS' features anyway). So the default settings are -f CDS -e gene,misc_feature -i locus_tag.

Outputs
The script prints to stdout details of what it is doing at each stage, and the names of all output files.

Naming conventions are:
1. A copy of the original table, excluding outgroup-variable SNPs, non-variable SNPs and excluded strains will be printed. The name of the file will include the initial prefix, plus:
- if outgroups are specified (-o), '_Xoutgroups' will be added to the file prefix (where X=number of outgroups)
- if a subset of strains is provided (-l), '_Xstrains' will be added to the file prefix (where X=number of strains included)
- '_var' will then be added to the prefix to indicate the subsequent files exclude non-variable SNPs

So, if you specified -p klebsiella -o outgroup1,outgroup2 -l strainlist.txt the output table would be named klebsiella_2outgroups_20strains_var.csv

The prefix is updated to 'klebsiella_2outgroups_20strains_var', so that all subsequent files (alignments, further filtering, trees, etc) carry this informative name.

2. Following this, each module extends the current prefix to tell you what the module did, and prints new output files.

aln: prints to [prevPrefix].fasta (does not update the file prefix)

filter: adds '_regionFiltered' to the existing prefix, prints new CSV table [prevPrefix]_regionFiltered.csv

cons: adds '_consXX' to the existing prefix, where XX is the proportion provided via -c

clean: adds '_cleanPxxWxy' to the existing prefix, where xx is the SNP pair window (provided via -P) and xy is the window for three of more SNPs (provided via -W)

core: adds '_coreXX' to the existing prefix, where XX is the proportion provided via -Z

vcf: normally prints to [prevPrefix].vcf, unless the PASS/FAIL vcf is requested via -A, then compound vcf prints to [lastPrefix]_gnr.vcf  (does not update the file prefix)

## Troubleshooting a Pipeline Run.

The RedDog pipelines works by send a series of jobs to be processed. When all the jobs in a stage are completed, the pipeline will then launch then next stage (or stages where the pipeline branches). If on attempting to launch jobs in this stage, it is found that a job (or jobs) failed during the previous stage, the pipeline will report an error – in most cases you will have to look back to see at which stage the first error occurred.

Note that the pipeline will continue to run after reporting this error, waiting for other jobs already launched to finish, however, no new jobs will be launched. On most occasions the pipeline will become stuck in this "listener" mode, so once you are sure all the jobs have really finished ('`showq`' using a separate login session will indicate when these have finished) the pipeline can then be halted using '`Ctrl-c`' (you may have to do this twice to get back to the command prompt).

If you know the stage the pipeline failed at, finding out why is relatively simple. In the RedDog folder, there will be a sub-folder called '`log`' (as long as the defaults haven't seen changed). Within this log folder are the '`stderr`' and '`stdout`' files for each job that has been run by the pipeline. You can look through the 'stderr' files for those from the stage with the failed job(s). By opening them you should be able to locate the associated error message. (If you are using a GUI file manager *e.g.* Cyberduck, look for those modified later with (usually) a larger file size.)

Note: the stderr and stdout files names have the following etology - <stage_name>.<job_number>.stdout (or stderr)

If the job was killed by the system due to walltime or memory allocation, see 'Large Data Sets, Large Read Sets and High Variation' above. If the error is not due to either of these two, you should check to see if the issue is a know one on the RedDog 'Issues' page on Github (check both open and closed issues). If you then think the error is a new error, or cannot see an obvious solution in the previous issue reports, you should then report it on the Github 'Issues' page.

See 'Reporting a Problem with RedDog' towards the beginning of this manual for the link to the RedDog Issues page, and how to report an error.

With runs where a large number of jobs are being executed, finding the file in the log folder with the error can prove difficult. To this end, there is a 'bash' script included in the RedDog folder, '`errorcheck.txt`', written by one of the alpha-testers of RedDog. To run '`errorcheck.txt`', first '`cd`' to the RedDog folder with the log folder you wish to search. Then enter '`./errorcheck.txt`' and the script will immediate launch.

# Pipeline Change History

**Current version:**

V1.0b Any fixes from final testing (DE)
Includes handling of "." in replicon name (DE)
Add licensing information to all scripts (DE)
one more post analysis script added [for Gubbins recombination analysis] (DE)
update parseGenContent.py (P/A matrix based on cover and depth) (DE)

**Previous versions:**

V0.1
converted to vcf output via mpileup instead of depreciated pileup (DE)
...
V0.2
tested version V0.1.1 (DE)
V0.2.1
added statistic reporting and fixed "success" handling (DE)
V0.3
tested version of V0.2.1 (DE)
V0.3.1
added alternative output paths to aid clean up (DE)
new name: pipe_VariantDiscovery.py (DE)
added q20 and q30 mpileups and associated stats collection (DE)
various cleanup of code/naming conventions used in pipeline (DE)
V0.3.2
added alternative path for IT or PE data analysis (DE)
V0.3.2.1
added alternative path for SE data analysis (DE)
added minimum read of depths option for variant filtering (DE)
V0.3.2.2
update in various tools (bwa, bamtools and tmap) (DE)
changes only affects config file
V0.3.3
tested version of V0.3.2.2 (DE)
V0.3.4
added options for qc of IT reads (DE)
V0.3.4.1
added size option for qc of IT reads (DE)
updated statistics reporting - minimum reads (DE)
version numbers change (1.x to 0.x) (DE)
V0.3.4.2
added pass/fail to stats reporting (DE)
added outgroup/ingroup to stats reporting (DE)
V0.3.5
tested version of V0.3.4.2 (DE)
V0.3.5.1
update to various tools (BWA and tmap) (DE)
output folders now created within the pipeline (DE)
added separate folder for success files within the temp folder (DE)
added two new output folders (bam and vcf) (DE)
V0.3.5.2
tested version of V0.3.4.1 (DE)
V0.3.5.3
changed to pipe_vda including:
allow for merging of new sets of reads into a prior run (DE)
inclusion of analysis pipelets (DE)
- pipe_VCFAnalysis and pipe_AllGeneCover
clean up of temp directory and/or output directory (if merging) (DE)
changed "type" to "readType" (DE)

slight change to pipeline order (DE)

V0.4

tested version of V0.3.5.3 (DE)

V0.4.0.1

merging of bams from different read sets of same strain (DE)
(either during "new run" or "merge run")

fixed bug in gene cover and depth matrices script (DE)

V0.4.0.2

tested version of V0.4.0.1 (DE)

V0.4.0.3

removal of QC from within pipeline (and testing) (DE)

V0.4.0.4

replace filter.awk with python-based filtering of all hets from Q30 vcfs (DE)
includes counting removed het SNPS and reporting same
in stat.tab (and testing) (DE)

V0.4.0.5

inclusion of parseSNPtable script (KH, DE)
and tree generation (DE)

V0.4.0.6

corrections to many scripts, including allele matrix calling
and downstream effects to pipeline (DE)
allele matrix calling now uses consensus sequences (DE)
addition of differences of SNPs as distance matrix in NEXUS format (DE)
gene cover and depth matrices no longer contain "fails" (DE)
addition of parseGeneContent script (KH, DE)
removal of q20 vcfs (and reporting) - not required (DE)

V0.4.0.7

removed duplicated stages from config file (DE)

V0.4.0.7.1

fix to allow sequences from different folders in the same run (DE)

V0.4.0.7.2

fix to deriveStats that let some failed reads pass on depth (DE)

V0.4.1

Map to reference with multiple "chromosomes": pangenome mapping (DE)
- simplest case: new run (no merging of runs or samples)
- up to stats collection ('collateRepStats', no post-stats analyses)

add final '/' to output path(s) if missing (DE)

V0.4.2

Map to reference with multiple "chromosomes": phylogenetic mapping (DE)
- simplest case: new run (no merging of runs or samples)
- up to stats collection ('collateRepStats', no post-stats analyses)

added start-up message (DE)
changed reference entry from GenBank and Fasta formats to GenBank or Fasta format (DE)
- fasta reference generated from user GenBank reference

added pre-run checks including
- pairs of reads exist before starting PE analysis (DE)
- check for 'sequence' option - bad pattern entry (DE)
- valid run and read types are entered (DE)

zeroing of SAM files when no longer needed (DE)

V0.4.3

added 'post-stats' analyses - pangenome and phylogeny - no Genbank
added pre-run reporting and run start confirmation (DE)

V0.4.4

added 'post-stats' analyses - pangenome and phylogeny - with Genbank

V0.4.4.1

various small fixes (DE)

V0.4.4.2

more various small fixes (DE)

V0.4.4.3

increased speed of deriveAllRepGeneCover and getCoverByRep (DE)

V0.4.4.4

conversion of pipeline to use SLURMed Rubra (DE)

V0.4.4.5

fix for bug in BWA sampe/samse v0.7.5 (DE)

V0.4.5

renamed pipeline (DE)
add merging of runs for pangenome and phylogenetic mapping (DE)
remove single replicon run (DE)
added bowtie2 to mapping options (all read types) (DE)
removal of tmap (DE)
removal and replacement of bamtools (pileup for coverage instead) (DE)
cleanup of pipeline scripting (amalgamation of repeated stages) (DE)
converted emboss call to a biopython script (DE)
add 'check_reads_mapped' variable for multiple replicon runs (DE)

V0.4.5.1

fix for replicon statistics generation for pangenome runs (DE)

V0.4.5.1.1

fix for all statistics generation when no reads map (DE)

V0.4.5.2

check that replicons all have unique names (DE)
check that output and out_merge_target folders are different (DE)
check that output folder is not empty string (DE)
splitting of getRepAlleleMatrix to improve performance (DE)
includes sequence list generation (start of post-run report)

V0.4.6

update to newer version of parseSNPtable.py (DE)
- generation of variable and conserved SNP tables (DE)
- includes of additional option of setting conservation level (DE)

further early checks that include:
- name of reference/replicons/isolates won't confuse post-NEXUS analysis (i.e. no '+')

fix for when a replicon consensus fasta is missing (DE)
includes new 'warning' file (DE)
change behaviour of outgroups - reported (also in outgroups.txt fle) but not removed (DE)
Editing and reorder of options in config file (DE)

V0.4.7

changed 'sequence_list.txt' generation to function (DE)
added stage counts and check before firing last stage deleteDir (DE)
added check for isolates/reads with identical names (DE)
fixes for errors in mergeRepStats and parseSNPtable (DE)
- latter includes fixes in script to improve performance (DE)

V0.4.8

include post-run report file function (DE)
test for 'output' folder prior to run (DE)
default conservation changed to 0.95 (DE)
consolidated chrom_info functions into pipe_utils (DE)
further replicon name checking (DE)
fix for pipe-generated gene 'tags' when missing (DE)
write cns warning files to outMerge on merge run (DE)

v0.4.9

split location of intermediate files to improve stability for large runs (DE)
changed -X switch in bowtie2 PE mapping from 500 (default) to 1500 (DE)
removal of some redundant scripts and stages in config file (DE)

V0.5

added check for deletion of previous run success file on merge run (DE)
added checkpoints for better pipeline running - will halt on errors as expected (DE)
includes changes to complex stages - flagFiles behaviour (DE)

V0.5.1

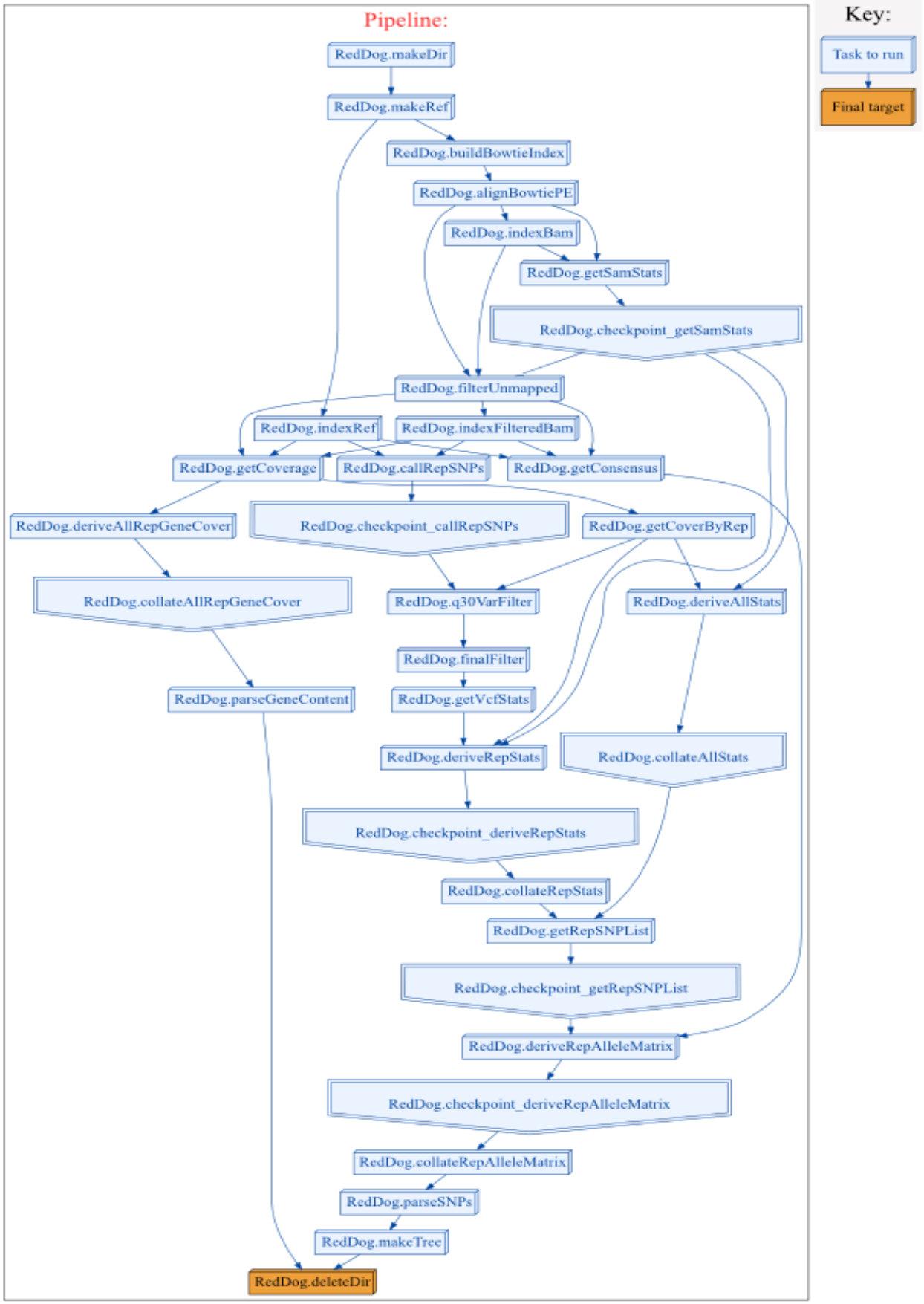
added -X option for bowtie2 mapping (DE)

fixed parseGeneContent output (DE)
updated to fixed and extended parseSNPtable (DE)
added scripts for tutorial (filterCoords.py, get_cover.py, getRecomb.R, plotTree.R) (DE)
changed getDifferenceMatrix to optional output in pipe, changed script to take options (DE)
added option for VCF output of filtered hets (DE)
implemented changes to run report from user feedback (DE)

V0.5.2 run report provides settings for merge runs (continuity checks) (DE)
includes more robust 'check_reads_mapped' (DE)
update to use SAMtools v1+ (DE)
includes (limited) addition of multiallelic SNP calling option - bcftools (DE)
added checkpoint_getSamStats to capture failure during initial BAM construction (DE)
changed bams from glob call to list call (DE)
various small fixes to some default values (DE)
fixed getRecomb.R (DE)

## Example Pipeline Flowchart

Figure 1. **Example Pipeline Flowchart** for a 'new' run using Illumina paired-end reads.



## Understanding the (more complex) Output - Examples

The following descriptions use the output files from the O104:H4 tutorial as examples. (See RedDog Tutorial. Some output lines have been edited to improve readability)

a) **Run statistics - summary**: <reference>_AllStats.txt
`2011C-3493_full_AllStats.txt` (header and first entry)

```
Isolate    Cover%_CP003289 Cover%_CP003290 Cover%_CP003291 Cover%_CP003292
     Depth_CP003289   Depth_CP003290   Depth_CP003291   Depth_CP003292
     Mapped%_CP003289 Mapped%_CP003290 Mapped%_CP003291 Mapped%_CP003292
     Mapped%_Total    Total_Reads      Insert_Mean      Insert_StDev
     Length_Max Base_Qual_Mean   Base_Qual_StDev
     A_%   T_%   G_%   C_%   N_%

11-4632_C3 99.3629360507    99.7368539935    97.2782516135    98.9670755326
     37.0178053113    36.2147637327    35.9647353768    19.8708414873
     97.0259499571    1.62915023095    1.31985026242    0.02309954955
     99.99805   2000000    499.9758   33.2385
     100   40.0000    0.0000
     24.6243    24.7370    25.2986    25.3391    0.0011
```

Reports on all replicons found in the reference file

Cover%_<replicon>: percentage of bases of the reference with at least one read mapped.
Depth_<replicon>: average depth of reads for bases with at least one read.
Mapped%_<replicon>: percentage of the total reads mapped to each replicon
Mapped%_Total: percentage of the total reads mapped to any replicon
Total_Reads: total number of reads (mapped and unmapped)
Insert_Mean (and _StDev): estimated size of the gap between paired end reads
Length_Max: longest read length
Base_Qual_Mean (and _StDev): average quality scores for the read set
A_%, T_%, G_%, C_%, and N_%: percentage of each nucleotide in the read set

b) **Run statistics by replicon**: <reference>_<replicon>_RepStats.txt
`2011C-3493_full_CP003289_RepStats.txt` (header and first entry)

```
Isolate Cover%_CP003289 Depth_CP003289 Mapped%_CP003289 Mapped%_Total
     Total_Reads      SNPs  Hets_Removed     Indels     Ingroup/Fail
11-4632_C3 99.3629360507    37.0178053113    97.0259499571    99.99805   2000000
     88    16    3     i
```

Report for each replicon in the reference (phylogeny run), or specified replicon(s) (pangenome)

Cover%_<replicon>: percentage of bases of the reference with at least one read mapped.
Depth_<replicon>: average depth of reads for bases with at least one read.
Mapped%_<replicon>: percentage of the total reads mapped to each replicon

Mapped%_Total: percentage of the total reads mapped to any replicon
Total_Reads: total number of reads (mapped and unmapped)
SNPs: total SNP count (does not include conservation filtering)
Hets_Removed: number of heterozygous calls filtered from the SNP set (High count *c.f.* SNPs could indicate contamination)
Indels: Number of indel variants called
Ingroup/Fail (pangenome) or Ingroup/Outgroup/Fail (phylogeny): Any isolate that fails one or more filters (percentage cover, depth or percentage mapped) will be set to 'f'. Otherwise they will be set to 'I' (ingroup) unless (for phylogeny runs) the number of SNPs is greater than 2 SD of the mean SNP count for all isolates that did not fail

c) **SNP allele matrix:** <reference>_<replicon>_alleles_var[_cons0.95].csv

```
Pos,Ref,11-02030,11-02033-1, … Ec11-5538,Ec11-5603,Ec11-
6006,GOS1,GOS2,H112180280,H112180282,H112180283,H112180540,H112180541,L
B226692,ON2011,TY-2482
9878,G,G, … C,G,G,G,G,G,G,G,G,G,G
15186,C,C, … C,C,C,C,C,C,C,G,C,C,C
```

Pos: position of SNP in reference sequence
Ref: SNP call in reference sequence

d) **Percentage cover by gene**: <reference>_CoverMatrix.csv

```
replicon__gene,01-09591,04-8351, …
H112180540,H112180541,LB226692,ON2011,TY-2482,ON2010
CP003289__O3K_00005,100.0,100.0, … 100.0,100.0,100.0,100.0,100.0,100.0
CP003289__O3K_00010,100.0,100.0, …
95.0090744102,100.0,100.0,100.0,100.0,100.0
```

replicon__gene: source of the gene; reference sequence and gene tag separated by '__'. If the genes in the GenBank file do not have locus tag, a tag based on the gene's position will be used (these are not added to the Genbank file!).

Each cell in the matrix gives the percentage cover of the gene (for bases with at least one read). The DepthMatrix.csv has exactly the same format, except cell values are the average depth of reads for the gene (for bases with at least one read). The PresenceAbsence.csv also has the same format, but cell values are either '1 'for present genes (coverage >= 95%) or '0' for absent (<95%).

e) **SNP consequences**: <reference>_<replicon>_alleles_var[_cons0.95]_consequences.txt

```
SNP  ref  alt  change    gene ancestralCodon derivedCodon   ancestralAA
     derivedAA product   ntInGene  codonInGene    posInCodon
9878 G    C    ns   O3K_25622 CTC  GTC  L    V    SogL protein   1516
     506  1
15186     C    G    ns   O3K_25647 CGG  CCG  R    P
     lipopolysaccharide core heptose(II)-phosphate phosphatase   548
     183  2
```

For each SNP called in the SNP allele matrix, there will be an entry in the consequences table; these consequences are also added to a new version of the GenBank file for the reference sequence. A ‘change’ can be synonymous (s), non-synonymous (ns) or intergenic. If the SNP occurs in a gene, further information is provided (as shown).