iGUIDE Documentation

Release v1.1.1

Christopher Nobles

Apr 17, 2025

CONTENTS:

iGUIDE is a software pipeline for processing and analyzing double-strand DNA break events. These events may be induced, such as by designer nucleases like Cas9, or spontaneous, as produced through DNA replication or ionizing radiation. A laboratory bench-side protocol accompanies this software pipeline, and can be found (https://doi.org/10.1186/s13059-019-1625-3).

This documentation gives the reader an overview of the pipeline, including how to install and process a sample dataset. Processing a sample data set is broken into a few parts:

1. Developing a configuration file and sample information
2. Setting up a run directory and acquiring the sequence data
3. Initializing the pipeline and understanding the output

After processing sequencing run(s), additional evaluation and analysis can be performed by supplying supplemental data and combining specimen outputs from several runs.

To get started, see *Quickstart Guide*!

CHAPTER

ONE

QUICKSTART GUIDE

Contents

1.1 Install

To install iGUIDE, simply clone the repository to the desired destination.:

```
git clone https://github.com/cnobles/iGUIDE.git
```

Then initiate the install using the install script. If you would like the installed environment to be named something other than 'iguide', the new conda environment name can be provided to the `install.sh` script as provided below.:

```
cd path/to/iGUIDE
bash install.sh
```

Or:

```
cd path/to/iGUIDE
bash install.sh -e {env_name}
```

Additionally, help information on how to use the `install.sh` can be accessed by:

```
bash install.sh -h
```

1.2 Setup a Run

Once the config and sampleInfo files have been configured, a run directory can be created using the command below where {ConfigFile} is the path to your configuration file:

```
cd path/to/iGUIDE
iguide setup {ConfigFile}
```

The directory should look like this (RunName is specified in the ConfigFile):

```
> tree analysis/{RunName}
analysis/{RunName}/
├── config.yml -> {path to ConfigFile}
├── input_data
├── logs
├── output
├── process_data
└── reports
```

Components within the run directory:

- config.yml - This is a symbolic link to the config file for the run
- input_data - Directory where input fastq.gz files can be deposited
- logs - Directory containing log files from processing steps
- output - Directory containing output data from the analysis
- process_data - Directory containing intermediate processing files
- reports - Directory containing output reports and figures

As a current convention, all processing is done within the analysis directory. The above command will create a file directory under the analysis directory for the run specified in by the config ('/iGUIDE/analysis/{RunName}'). At the end of this process, iGUIDE will give the user a note to deposit the input sequence files into the /analysis/{RunName}/input_data directory. Copy the fastq.gz files from the sequencing instrument into this directory if you do not have paths to the files specified in the config file.

iGUIDE typically uses each of the sequencing files (R1, R2, I1, and I2) for processing since it is based on a dual barcoding scheme. If I1 and I2 are concatenated into the read names of R1 and R2, it is recommended the you run `bcl2fastq ... --create-fastq-for-index-reads` on the machine output directory to generate the I1 and I2 files.

As iGUIDE has its own demultiplexing, it is recommend to not use the Illumina machine demultiplexing through input of index sequences in the SampleSheet.csv. If your sequence data has already been demultiplexed though, please see the *User Guide* for setup instructions.

1.3 List Samples in a Run

As long as the config and sampleInfo files are present and in their respective locations, you can get a quick view of what samples are related to the project. Using the `iguide list_samples` command will produce an overview table on the console or write the table to a file (specified by the output option). Additionally, if a supplemental information file is associated with the run, the data will be combined with the listed table.:

```
> iguide list_samples configs/simulation.config.yml
```

(continues on next page)

(continued from previous page)

```
Specimen Info for : simulation.

 specimen   replicates       gRNA        nuclease
---------- ------------ --------------- ----------
   iGXA         1            TRAC         Cas9v1
   iGXB         1        TRAC;TRBC;B2M    Cas9v1
   iGXD         1             NA            NA
```

1.4 Processing a Run

Once the input_data directory has the required sequencing files, the run can be processed using the following command:

```
cd path/to/iGUIDE/
iguide run {ConfigFile}
```

Snakemake offers a great number of resources for managing the processing through the pipeline. I recommend familiarizing yourself with the utility (https://snakemake.readthedocs.io/en/stable/).

1.5 An Example Workflow

To perform a local test of running the iGUIDE informatic pipeline, run the below code after installing. This block first activates your conda environment, 'iguide' by default, and then creates a test directory within the analysis directory. The run information is stored in the run specific configuration file (config file). Using the `-np` flag with the snakemake call will perform a dry-run (won't actually process anything) and print the commands to the terminal, so you can see what snakemake is about to perform. Then the entirety of processing can start.:

```
# If conda is not in your path ...

source ${HOME}/miniconda3/etc/profile.d/conda.sh

# Activate iguide environment

conda activate iguide

# After constructing the config file and having reference files (i.e. sampleinfo)
# You can check the samples associated with the run.

iguide list_samples configs/simulation.config.yml

# Create test analysis directory

iguide setup configs/simulation.config.yml

# Process a simulation dataset

iguide run configs/simulation.config.yml -- -np
iguide run configs/simulation.config.yml -- --latency-wait 30

# Processing will complete with several reports, but if additional analyses are required,
# you can re-evaluate a run by its config file. Multiple runs can be evaluated together,
# just include multiple config files.
```

(continues on next page)

(continued from previous page)

```
iguide eval configs/simulation.config.yml \
  -o analysis/simulation/output/iguide.eval.simulation.test.rds \
  -s sampleInfo/simulation.supp.csv

# After evaluation, generate a report in a different format than standard.
# Additionally the evaluation and report generation step can be combined using
# config file(s) as inputs for the 'report' subcommand (using the -c flag instead of -e).

iguide report -e analysis/simulation/output/iguide.eval.simulation.test.rds \
  -o analysis/simulation/reports/report.simulation.pdf \
  -s sampleInfo/simulation.supp.csv \
  -t pdf

# When you are all finished and ready to archive / remove excess files, a minimal␣
↪structure
# can be achieved with the 'clean' subcommand.

iguide clean configs/simulation.config.yml

# Or you realized you messed up all the input and need to restart

iguide clean configs/simulation.config.yml --remove_proj

# Deactivate the environment

conda deactivate
```

1.6 Reviewing Results

The output reports from a run are deposited under `analysis/{RunName}/reports`. For more informtion on output files, see *User Guide*!

CHAPTER

TWO

USER GUIDE

Contents

2.1 Nomenclature and Semantics

Before diving too far into the documentation, it is important to understand some of the nomenclature and semantics used throughout this documentation. Focus will be mostly spent on words important to the workflow and that may be ambiguous to when used without definition. Some of these words may seem like they overlap in definition, and in certain situation, they do. In these situations, we should still give proper designation to each catagory to distiguish them from each other.

2.1.1 The three S's

We'll focus first on the three S's that relate to what we are working with, Subject, Specimen, and Sample.

- Subject: the who or what we are working with, this could be a patient or an experiment. It is important to remember which of the downstream identifiers are associated with a specific subject.
- Specimen: is collected from a subject and is the start of the protocol. A specimen for iGUIDE could be considered a tube of starting gDNA. This will be the actual material that will be worked with for the protocol.
- Sample: While many people commonly use specimen and sample interchangibly, here we note that a sample comes from a specimen. We make this distiction because we realize there are multiple ways to workup a single specimen, each of these different ways is a different sample. Samples are taken from the specimen, just like the specimen is taken from the Subject.

In the following workflow, you'll notice that in certain places, we refer to 'sampleName' (such as in the sampleInfo file), or 'specimen' (such as in the supplemental data file). These designations are consistent with the above definitions and it is expected that the user will follow these customs.

How do we destinguish Subject, Specimen, and Sample? During processing these identifiers will need to be distiguished from each other using different nomenclature. Below is an example of a naming scheme for the three identifiers.:

```
Subject   Specimen    Sample
{patient} {Spec.ID}   {Spec.ID-info-rep.ID}
pND405    iGSP0015    iGSP0015-neg-1
pND405    iGSP0015    iGSP0015-neg-2
pND405    iGSP0015    iGSP0015-neg-3
```

Here we have an example workflow. Subject identifiers are not usually part of the processing, we consider Subject typically during data interpretation through reviewing the output reports. Subject identifiers can be included in supplemental files with runs. Specimens can have identifiers (or IDs). For iGUIDE, it is easiest to use a single alpha-numeric string as an identifier (without delimiters!).

Following the above practice, the specimen ID can be included in a sample ID (or sampleName) along with additional information. As indicated above, iGUIDE will treat sampleNames as three part strings, the specimen ID is at the beginning, delimited (or separated by a "-") from additional information. The last part of the string is a replicate identifier, expected to be numeric. In practice, we find it best to create 4 samples for processing from a single specimen. This limits the possibility for PCR jackpotting an allows an analyst to utilize capture-recapture statistics for population estimation. The remainder of the string that is not captured in the first or last components is not directly used by iGUIDE, except as a unique identifier of the specimen. Therefore it is a great place to indicate sample specific treatments.

Given the above example, three different samples have been indicated, all from a single specimen and single subject. During processing, the user will indicate each of sample in the sampleName column of the sampleInfo file. When

iGUIDE returns the analysis, each specimen will be indicated. So while three samples go into the pipeline, data will be combined in the output to represent the single specimen.

More information can be found about specimen and sampleNames in this user guide.

2.1.2 The Experiment

While writing this documentation, I thought it would be helpful to explain in a general sense what an experiment might look like using with repective terminology of this software.

For a respecitive subject (patient, individual experiment, ...) that has been treated with the marker dsODN during genome editing, specimens are harvested from various conditions (with nuclease, with different targets controlled by gRNAs, ...). This harvesting yeilds genomic DNA which is commonly cataloged into a specimen database holding metadata and experimental parameters for the different specimens.

Samples are then taken from these specimens, typically 4 samples (see protocol from iGUIDE manuscript), and processed through the iGUIDE protocol. Before sequencing, a sampleInfo sheet would be constructed where each row of a csv file indicates a different sample that was processed along with the samples barcode and demultiplexing information.

During sequencing (or after), a run specific configuration file (config file) would be constructed by one or two parties. There is run specific information that needs to be included, such as: target sequence patterns, nuclease profiles, treatment information, etc. If a variable changes throughout the samples, then it can be indicated in the sampleInfo file, while if it is constant, it can be indicated in the config file.

The latter part of the config is reviewed and checked by the individual who will computationally process the run. This portion of the config file contains parameters that modify or tune the software to run on different systems.

After the computational processing has completed, a stat report and analytical report are generated in the reports directory. These can be reviewed by respecitive parties.

Additionally, if multiple runs contain samples to be analyzed together, auxiliary commands in iGUIDE allow for the computational analyst to generate new reports combining multiple sequencing runs together.

If the user is unsure if the experiment or would work with this type of analysis, feel free to contact the maintainers of iGUIDE.

2.2 Subcommands

Once installed, iGUIDE utilization is broken down into subcommands as indicated in the Figure 1 below. A description of these commands are reviewed here to give the user an understanding of how the software would work from a workflow view point.

Primary subcommands: Used for standard or primary workflow of processing sequencing runs.

- `setup` : This subcommand initiates a project or run directory. It requires a config file and will create the new project directory within the iGUIDE/analysis directory.
- `run` : This subcommand will process a run given a config file using a Snakemake workflow (https://snakemake.readthedocs.io/en/stable/). Therefore, Snakemake specific commands can be passed into the `run` subcommand. All Snakemake specific commands should come after a `--` break in the options.

Auxiliary subcommands: Used for auxiliary workflows which further dive into analyzing the processed data.

- `eval` : Short for evaluation, this subcommand will analyze a run's data and yeild an RDS file (R-based data file). Supplemental data can additionally be passed into the evaluation to group specimens together for analysis and include metadata. This output object has a host of broad analysis that are based in the input information.
- `report` : This will generate a full report on the given config file(s) or input evaluated RDS file. The report is defaultly produced as an html document but can be changed to a pdf if the correct latex libraries are installed.

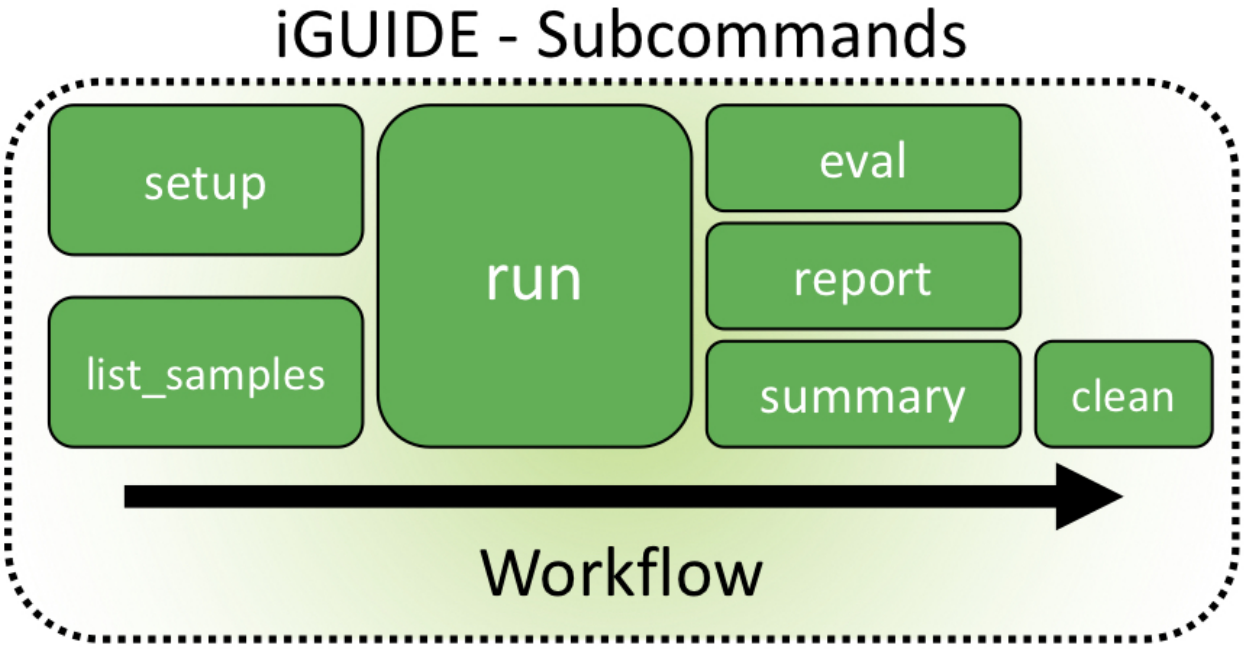

Fig. 1: Figure 1. iGUIDE Subcommands: setup, run, eval, report, summary, clean.

Additionally, all figures and tables can be output as independent files (pdf and png formats for figures and csv formats for tables).

- `summary` : Similar to the report but with some reduced utility, this subcommand will output a single text file that overviews the data. This is readable on the terminal and is helpful for getting quick answers to data questions if working on the command line.

Additional subcommands: Used for cleanup and helpful suggestions for processing.

- `clean` : After processing, most intermediate data files are removed as they are designated temparary, but other file still exist within the run directory that may inflate the size and are no longer needed, such as input data and log files. The `clean` subcommand will remove files no longer required. A "clean" run directory can still be used with `eval`, `report`, and `summary`. Additionally, this subcommand can remove the entire run directory by passing the `--remove_proj` flag.
- `hints` : Prints out a message with Snakemake option hints to help with using the `run` subcommand.

2.3 Workflows

A workflow is simply how data is moved from an unprocessed state (like sequencing data off an Illumina sequencer) to a processed state (a final report). Below we will review the primary and auxiliary workflows iGUIDE is designed to handle.

2.3.1 Primary Workflow

In the primary workflow, we consider how to get from input sequence information to processed reports. To initiate this process, the user needs to gather the information and complete two files, the configuration file (config file) and the sample information file (sampleInfo file). These two files will tell iGUIDE how to process the sequence information, sample specific parameters should be included in the sampleInfo file while constant parameters can be simply specified in the config file. Once these two files are completed, they can be deposited into their repective directories (config file

–> iGUIDE/configs and sampleInfo file –> iGUIDE/sampleInfo). Additionally, if a supplemental file (supp file) is to be included, it is easiest to deposit this file with the sampleInfo file, in iGUIDE/sampleInfo.

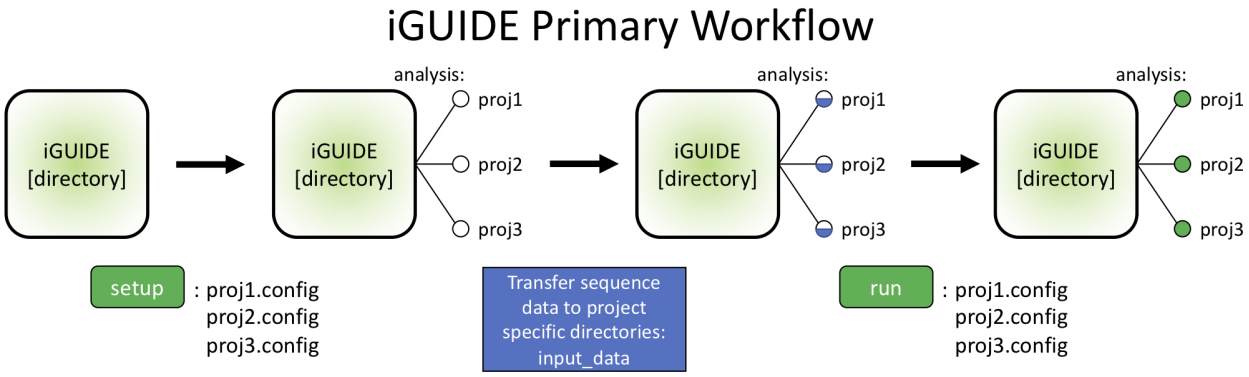

Fig. 2: Figure 2. Primary workflow for processing input sequencing files to processed runs with data deliverables like reports and figures.

With the config, sampleInfo, and potentially supp files in place, the user can use `iguide setup {path/to/[run].config.yml}` to create a new run directory. In Figure 1, three runs have been developed, named proj1, proj2, and proj3. Each of these would have a different config and sampleInfo file. With the files in their respective directories, the user would run `iguide setup configs/proj1.config.yml` to create the "proj1" run directory in the analysis directory, and then repeat the command with the other two config files to have a total of three empty run directories under the analysis directory.

Once the run directories are setup, the input data needs to be located. This can be done in a number of ways. In the config file, the user can specify the path to the sequence files (preferably not demultiplexed, see latter sections for skipping demultiplexing). The user can create symbolic links to the data within the input_data directory of the run directory, or the user can simply deposit the sequence files (fastq.gz) into the input_data directory.

With config file, sampleInfo file, and sequencing files ready, the user can start processing with `iguide run configs/{run}.config.yml`. Recall that the `run` subcommand is built on a Snakemake workflow, so additional Snakemake options can be passed after `--` when issuing the command. For example, `iguide run configs/proj1.config.yml -- --cores 6 --nolock -k`, tells Snakemake to use 6 cores for processing, do not lock the working directory (helpful for running multiple processing runs at the same time), and keep going even if one job has an error.

Allowing the `iguide run` command to go to completion will yeild a processed data run. At this point, if calling the same "run" command on a project, Snakemake should return a message indicating that there is nothing to do. If for some reason processing gets terminated, `iguide run` and Snakemake will pickup from where it left off in the processing.

If the user is content with the processing, then they can run the `iguide clean` command to clean up a specific run directory (shown in Figure 3 below). This leaves the output data (useful in the auxiliary workflow) and the reports, but will remove input_data and log files. Additionally if the user wants to remove the run directory completely, they can also use the `iguide clean` command with an optional flag.

2.3.2 Auxiliary Workflow

After running the primary workflow on several runs, or if the user would like to change specific parameters (gene lists, target sequences, . . .) then the auxiliary workflow becomes quite useful.

There are three subcommands included in this workflow: `eval`, `report`, and `summary`. Each of them work in similar ways, but have different outputs.

The `iguide eval` is a focal point of the auxiliary workflow. This command will process one or more runs and analyze them in a consistent manner, so the user is confident they don't have a mixed data set. This subcommand will output

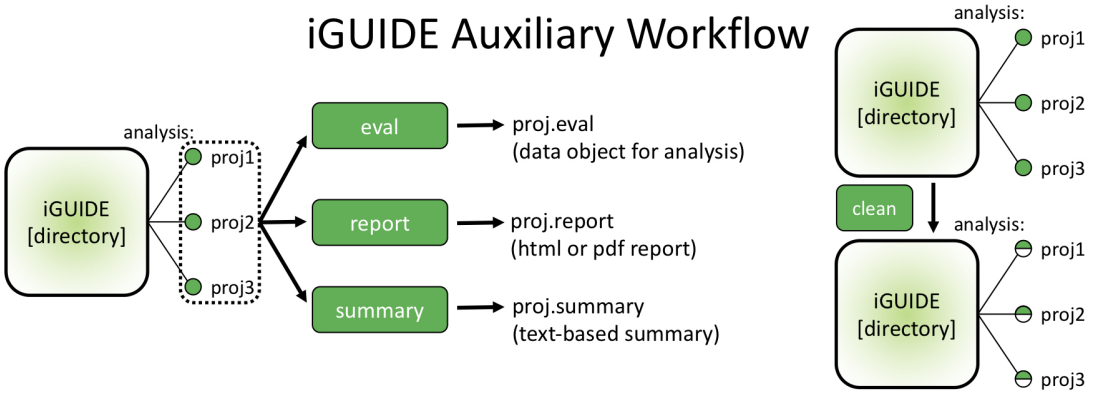

Fig. 3: Figure 3. Auxiliary workflow helps with subsequent analysis of the processed data.

a binary R-based file (*.rds) which can be read into an R environment with the function base::readRDS(). This file contains a host of analysis and can be used with the other two subcommands, `report` and `summary`.

The `iguide report` will output an html or pdf analysis of the evaluated dataset. This is the standard deliverable from the iGUIDE package. Additionally, the command can generate the figures and tables along with the report. `iguide summary` is very similar, but only generates a text-file based report. Both will take `eval` output files as an input, but they can also be used with the same input as would be given to `eval`, config file(s).

Supplemental files carrying specimen-based metadata can also be included in the auxiliary commands. Any specimen not indicated in the supp file will be dropped from the analysis. This means the user can select which samples are included in the analysis by specifying the associated specimens to include, even if the specimens are across multiple runs.

With this knowlege in hand, the remainder of the documentation should have more context as to how it is applied to processing data with the iGUIDE software.

2.4 Requirements

- A relatively-recent Linux computer with more than 2Gb of RAM

We do not currently support Windows or Mac. (iGUIDE may be able to run on Windows using the [WSL](https://docs.microsoft.com/en-us/windows/wsl/about), but it has not been tested).

2.5 Installing

To install iGUIDE, simply clone the repository to the desired destination.:

```
git clone https://github.com/cnobles/iGUIDE.git
```

Then initiate the install using the install script. If the user would like the installed environment to be named something other than 'iguide', the new conda environment name can be provided to the `install.sh` script as shown below.:

```
cd path/to/iGUIDE
bash install.sh
```

Or specify a different environment name.:

```
cd path/to/iGUIDE
bash install.sh -e {env_name}
```

Additionally, help information on how to use the `install.sh` can be accessed with the `-h` flag.:

```
bash install.sh -h
```

2.5.1 Testing

If the user would like to run a test of the software during the installation, the install script has a `-t` option that helps with just that. The below command will install the software with the environment named 'iguide' and test the software with the built-in simulated dataset during installation. Be ready for the testing to take a little bit of time through (up to 30 mins or so).:

```
bash install.sh -e iguide -t
```

Otherwise, the testing can be initiated after install using the following command.:

```
bash etc/tests/test.sh {env} {cores}
```

Where `{env}` would be the environment the user would like to test, "iguide" by default, and `{cores}` would be the number of cores to run the test on. The test will complete faster given more cores.

The test dataset can be regenerated with a script provided in the iGUIDE/etc/tests/construct_scripts directory, `simulate_incorp_data.R`. This script is configured by a partner config.yml file, `sim_config.yml`. A quick look through this configuration and the user can change the size of the simulated data output, rerun the script to generate new data, and develop a new test for iGUIDE.:

```
cd etc/tests/construct_scripts
Rscript simulate_incorp_data.R sim_config.yml
```

There are two scripts included in the tools/rscript directory that work with the simulated data. The first is designed to check the accuracy compared to the "truth" dataset that the simulated data was built on. To run that script, follow the command below.:

```
Rscript tools/rscripts/check_test_accuracy.R configs/simulation.config.yml etc/tests/
↪Data/truth.csv -v
```

The second script checks output files by their md5 digest, therefore any changes to the test (including generating new data, changing the aligner, changing parameters, ...) could make the test fail.:

```
Rscript tools/rscripts/check_file_digests.R etc/tests/simulation.digests.yml -v
```

Both testing scripts will exit with exit code 1 if they fail, which makes them easy to build into integration testing.

2.5.2 Updating

Over time, components of iGUIDE will be updated, including environmental builds, the commandline interface (python library or lib), and the supporting R-package (iguideSupport or pkg), as well as the standard code base. To update these, pull the latest release from GitHub with the following command after installation.:

```
git pull origin master
```

Once this has updated, the user should update their install by running the install script with the update option.:

```
bash install.sh -u all
```

It is recommended to update everything if the user is unsure of what has been updated. If the user just wants to update specific parts of the software through, they can use `env`, `pkg`, or `lib` after the `-u` flag to specify a component.

It is recommened that after updating, the user rerun the testing scripts to make sure the software is working appropriately on the specified system.

2.5.3 Uninstalling

To uninstall iGUIDE, the user will need to remove the environment and the directory.

To remove the environment and channels used with conda:

```
cd path/to/iGUIDE
bash etc/uninstall.sh
```

Or:

```
cd path/to/iGUIDE
bash etc/uninstall.sh {env_name}
```

If the user would rather remove the environment created for iGUIDE, it is recommended to use conda. This will leave the channels within the conda config for use with other conda configurations:

```
conda env remove -n iguide
```

Or:

```
conda env remove -n {env_name}
```

To remove the iGUIDE directory and conda, the following two commands can be used:

```
# Remove iGUIDE directory and software
rm -r path/to/iGUIDE

# Remove conda
rm -r path/to/miniconda3
```

2.5.4 Manual Install

Installing miniconda Skip to installing iGUIDE if you already have miniconda or anaconda installed. These can be executed in your home directory.

Get the latest version of miniconda install script:

```
__conda_url=https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
wget -q ${__conda_url} -O miniconda.sh
```

Installing miniconda through downloaded script. You can choose a different path for the install, here it is installed into the home directory:

```
__conda_path="~/miniconda3"
bash miniconda.sh -b -p ${__conda_path}
```

Source conda to activate the installation for use:

```
source ${__conda_path}/etc/profile.d/conda.sh
```

Installing iGUIDE The following commands should be called from within the iGUIDE directory (*/path/to/iGUIDE/*) after the repository is cloned.

Install the conda environment from the requirements file. The name field here can be changed to what you would like to call the environment, default for the install script is 'iguide':

```
conda create --name=iguide --quiet --yes --file etc/build.b1.0.1.txt
```

After successful creation of the environment, activate the iguide environment (or what you've named it):

```
conda activate iguide
```

Install the supporting R-package into the environment:

```
R CMD INSTALL tools/iguideSupport
```

Setup your environmental variables:

```
__iguide_dir=$(pwd)

echo -ne "#/bin/sh\nexport IGUIDE_DIR=${__iguide_dir}" > \
    ${CONDA_PREFIX}/etc/conda/activate.d/env_vars.sh

mkdir -p ${CONDA_PREFIX}/etc/conda/deactivate.d/

echo -ne "#/bin/sh\nunset IGUIDE_DIR" > \
    ${CONDA_PREFIX}/etc/conda/deactivate.d/env_vars.sh
```

You should now deactivate and reactivate the environment to initiate the environmental variables:

```
conda deactivate
conda activate iguide
```

Lastely, install the command line interface for iGUIDE using *pip*:

```
pip install --upgrade tools/iguidelib/
```

Testing Test to make sure the components were installed correctly.

Test for required R-packages installed:

```
$(Rscript tools/rscripts/check_for_required_packages.R &> /dev/null) && echo true ||␣
↪echo false
```

Check to make sure iguideSupport was installed correctly:

```
$(Rscript tools/rscripts/check_pkgs.R iguideSupport &> /dev/null) && echo true || echo␣
↪false
```

Run unit tests for iguideSupport:

```
`$(Rscript tools/rscripts/check_iguideSupport.R &> /dev/null) && echo true || echo false'
```

Check to make sure the pip install of the CLI was successful:

```
command -v iguide &> /dev/null && echo true || echo false
```

Run tests for iGUIDE, this step will take a little bit to complete if it starts processing the test case. Go grab a coffee, you deserve it getting to this point. :) The test will activate the environment as part of the test, so you should deactivate your environment first and then initiate the test:

```
conda deactivate
bash etc/tests/test.sh iguide
```

2.6 Config Files

Configuration files, or configs for short, contain both run-related and pipeline-related information. This is by design. For reproducibility it is easiest to have what was processed and how it was processed in the same location. There should be one config file for each sequencing run to be processed. Below is a brief summary of how to 'configure' your config file to your specific run.

Config files need to be named in the format '{RunName}.config.yml', where `{RunName}` is a parameter set within the config file for the run. For example, the default run configuration file is named `simulation.config.yml`, so the run name is `simulation`.

Config files can be deposited anywhere in the users directory, but a dediacted directory has been included in the release of iGUIDE. For convienence, config files can be placed in `iGUIDE/configs/`.

For sample specific information, input is more easily placed in a sampleInfo file. See the included section regarding sample info files.

2.6.1 File Layout

Config files are in a `yaml` format, but are broken into two parts. The first contains run specific information that should be filled out by an individual familiar with the sequence data used in the laboratory bench-side protocol. Additionally, they should be aware of the biochemistry related to the enzymes and sequences they are using.

The second part (below the divide `----`) should be filled out by an individual familiar with the bioinformatic processing. Explanations of the different portions can be found in the following pages.

2.6.2 Run Specific Information

Run configuration

`Run_Name`
: This is the name of the sequencing run, and should only contain alpha-numeric characters. Underscores (_) and dashes (-) are also allowed within the run name parameters. Other symbols should not be included, such as a dot (`.`). The run name is further used by the software to link files and directories together, so it will need to be consistent whenever it is used. Examples include: iGUIDE_190201_B6V99, 181213_PD1_T-cell_exp.

`Sample_Info`
: This is a file path to the sample information file. It can either be an absolute file path or relative file path. If the file path is relative though, it will need to be relative to the Snakefile used by the iGUIDE software. For more information about this file, please see the Sample Information page.

`Supplemental_Info`
: Similar to `Sample_Info`, this is a file path to a supplementary file which can contain information related to experimental parameters or patient information. This will be used during the report output, which will group samples with identical parameters. The format for this file is quite loose, and it only requires a single column `Specimen`, which should match the names of specimens in the sample information file. For more information about this file, please see the Supplemental Information page. If no file is to be used, set the value for this parameter to `"."` and make sure to set the `suppFile` in the run protion of the config to `FALSE`.

`Ref_Genome`
: This is a designation for the reference genome to used during processing. The genome will need to be included in the R libraries through BioConductoR prior to running the software. The human genome draft `hg38` is included by default. Please see information on the BioConductoR package 'BSgenome' for installing alternative genomes.

`Aligner`
: Options include either 'blat' or 'bwa', though at this time, only 'blat' is supported. Future versions of iGUIDE may support other alignment softwares. Please contact the maintainers if you have a favorite you would like to see listed here.

`UMItags`
: This is a logical parameter indicating whether to capture unique molecular indices (UMI) sequence tags ('TRUE') during processing. **Note:** Ambiguous nucleotides will need to be identified in the barcode fields of the sampleInfo file. Please see supplied simulated sampleInfo file for example.

`Abundance_Method`
: Options include 'Read', 'UMI', or 'Fragment' (default) for calculating the abundance method. 'Fragment' refers to the use of unique fragments lengths (see *SonicAbundance <https://doi.org/10.1093/bioinformatics/bts004>*) to quantify abundances of unique observations. 'UMI' will change the abundance method to use the unique molecular indices (**Note** that the `UMItags` option will need to be set to TRUE for this feature to work). 'Read' will change the abundance counts to read counts, yet this method may be unreliable due to PCR jackpotting or bias.

Sequence files

`Seq_Path`
: This is the file path to the sequence files. Rather than repeating the path for each below, just include the path to the directory containing the files.

`R1 / R2 / I1 / I2`
: These parameters should be the file names of the sequence files to be analyzed by the iGUIDE software. It is recommened to pass complete sequencing files to iGUIDE rather than demultiplexing prior to analysis.

SampleInfo formating

`Sample_Name_Column`
: This is the name of the column in the sample information file which contains identifiable information about samples. An appropriate format for the sample names is "{specimen}-{rep}" where 'specimen' is an alpha-numeric designator for the specimen and 'rep' is a numeric identifier for technical or biological replicates, separated by a dash (-). Replicates will be pooled during the final analysis, so if you want them to be separate in the report, make sure you give each specimen a different identifier. For example, iGSP0002-1 and iGSP0002-2, will be pooled together for the report and analysis, but iGSP0002-1 and iGSP0003-1 will not. These names will be used in naming files, so do not include any special characters that will confuse file managment. Try to stick to common delimiters, such as "-" and "_". A good practice is to put specimen identifiers at the beginning, replicate identifiers at the end following a "-", and anything else descriptive in the middle. For example, iGSP0002-neg-1, can specify the priming orientation the sample was processed with.

Sequence information

`R{1/2}_Leading_Trim`
: Sequence to be removed from the 5' or beginning of the R1 or R2 sequences. Commonly a linker or fixed sequence that is part of the priming scheme during amplification. If no sequence should be removed, just include `"."`. If the sequence is sample or specimen specific, it can be included in the sample information file and indicated in these fields as `"sampleInfo:{column}"`, where 'column' is the column name with the data in the sample information file.

`R{1/2}_Overreading_Trim`
: Similar to the `Leading_Trim` parameters, these parameters indicate the sequence that should be removed from the 3' or end of the reads if it is present. Again, if no sequence should be removed, use a `"."` or if the data is present in the sample information file, `"sampleInfo:{column}"`.

`R2_Leading_Trim_ODN`
: This is a key parameter difference between iGUIDE and the original GUIDEseq method. This parameter indicates the sequence that is part of the dsODN but is **not** primed against. This sequence should directly follow the `R2_Leading_Trim` sequence and should be a reverse complement of the beginning of the `R1_Overreading_Trim` sequence if the iGUIDE dsODN is being used. For GUIDEseq, simply include `"."`, or if you have multiple sequences, then specify in the sample information file as `"sampleInfo:{column}"`.

Target sequence information

`Target_Sequences`
: This parameter specifies the target sequences, **not including** the PAM sequences for guide RNAs. An acceptable input format would be `{target_name} : "{sequence}"` (i.e. `B2M.3 : "GAGTAGCGCGAGCACAGCTANGG"`) and additional target sequences can be included, one per line, and each indented at the same level. The input format of `{target_name} : {target_seq}` needs to be maintained for proper function. The 'target_name' in this situation will need to match the 'target_name' used in the `On_Target_Sites` and `Treatment` parameters. 'target_name' should follow a common format, and use standard delimiters, such as "-", "_", and ".". For example: `B2M.3`, `TRAC.1.5`, `TruCD33v5`.

`On_Target_Sites`
: This parameter indicates the specific location for editing by the target enzyme. There should be one line for each on-target site, even if there are more than one on-target sites for a given target sequence. Typically the input format should follow `{target_name} : "{seqname}:{+/-}:{position}"`, where 'target_name' matches the name of the given target sequence, and if multiple on-target sites exist, then the names can be expanded using a `{target_name}'#` notation. Additionally, the notation can be expanded to `{target_name} : "{seqname}:{+/-/*}:{min.position}-{max.position}"`, where '*' indicates either orientation and 'min.position' and 'max.position' represent the numerical range for the on-target site. The value for each on-target site specifies the location or genomic coordinates of nuclease activity. The 'seqname' indicates the chromosome or sequence name, an orientation of '+' or '-' is given to the location depending on the editing orientation (in line with positional numbering is '+' and opposite is '-', unknown or both is '*'), and the 'position' or 'min/max.position' indicates the nucleotide(s) of editing. For Cas9, the position of editing is commonly between the 3rd and 4th nucleotide from the 3' end of the targeting sequence (not including the PAM). Being off by a nucleotide or so will not cause any problems. Example below.:

```
On_Target_Sites :
  TRAC.5 : "chr14:+:22547664"
  TRBC.4'1 : "chr7:+:142792020"
  TRBC.4'2 : "chr7:+:142801367"
  PD1.3 : "chr2:-:241858808"
  TRAC.3.4 : "chr14:-:22550616-22550625"
  B2M.3 : "chr15:*:44711569-44711570"
  CIITA.15.1 : "chr16:+:10916399"
```

Specimen target treatment

`Treatment`
: This parameter indicates how samples were treated. If samples were all treated differently, then this information can be included in the sample information file as `all : "sampleInfo:{column}"` where 'column' is the name of the column with the information. If a single sample was treated with more than one target sequence, then delimit multiple target names by a semicolon (`;`), i.e. `all : "B2M;TRAC;TRBC"`. Additionally, each specimen can be indicated individually on a new line. Only specimen names should be given here and provided individually, not sample identifiers. This means that if your sample names follow the suggested format, "{specimen}-{replicate}", you would only specify the "{specimen} : {treatment}" underneath this parameter.

Specimen nuclease treatment

`Nuclease`

Similar to target treatment above, this parameter dictates which nuclease(s) where used on the specimens. This refers to the class of nuclease, such as Cas9 or Cpf1, which behave differently when they edit DNA. Notation can follow the same as above, if all specimens were treated with the same class of nuclease, then just specify 'all : "{nuclease_profile}"', or list out by specimen. Additionally you can specify the column in sampleInfo in the same format as above. Currently, iGUIDE does not support processing for specimens with multiple classes of nuclease profiles. Only one profile can be specified per specimen.

`Nuclease_Profiles`
See below section on nuclease profiles.

2.6.3 Processing Information

Below are parameters that are used to process the large amount of data, such as setting memory suggestions if resources are specified or parameters for sequence alignments. While these figues may not be relevant to the bench scientist, they are particulars for computational scientists.

Resource management is not required, but it can help when using HPC or limiting jobs. You are encouraged to spend some time optimizing if you would like, these parameters work out well on the designer's platform.

iGUIDE configuration

`Read_Types`
This parameter should include which read types will be used in the analysis, i.e. `["R1", "R2", "I1", "I2"]`. This follows a list notation is Python. If only single barcoding or some other method is employed and a read type is not included, simply leave it out of the example.

`Genomic_Reads`
This parameter is similar to the `Read_Types` but only indicates which reads contain genomic information rather than indexing.

`readNamePattern`
This is a regex pattern for which to gather read names, it should not make the read name sequencing orientation specific, R1 and R2 should have the same read name. The default works well for Illumina based readnames `[\w\:\-\+]+`. For R-based scripts to interpret the regex correctly, you will need to use double escapes, `[\\w\\:\\-\\+]+`.

Memory Management

`defaultMB / demultiMB / trimMB / filtMB / consolMB / alignMB / qualCtrlMB / assimilateMB / evaluateMB / reportMB`
Controls the amount of memory allocated to each of these processes during snakemake processing. While working on a server or multicored machine, these parameters will work internally to help schedule jobs. Each value will act as an upper limit for the amount of MB of RAM to expect the process to take, and schedule jobs appropriately using the `--resources mem_mb={limitMB}` flag with Snakemake. During HPC use, these parameters can be combined with the cluster config to schedule specific memory requirements for jobs. Additionally, if the `--restart-times {x}` is used where "x" is the number of times to restart a job if it fails, then the amount of memory for the job will increase by a unit of the parameter. For example, if a trimming job fails because it runs out of memory, then restarting the job will try to allocate 2 times the memory for the second attempt. All parameters should be in megabytes (MB).

Demultiplexing parameters

`skipDemultiplexing`
Logical (either TRUE or FALSE) to indicate if demultiplexing should be carried out. If FALSE, sequence files (*.fastq.gz) need to be placed or linked in the input_data directory of an existing project directory (as with `iguide setup`), one sequence file for each type (R1, R2, I1, I2). These need to be identified in the "Run" portion of the config file. If TRUE, then demultiplexed files need to be included in the input_data directory of an existing project

directory. The files need to be appropriately named, in the format of `{sampleName}.{readtype}.fastq.gz`, where `sampleName` matches the 'sampleName' column found in the associated 'sampleInfo' file, and `readtype` is R1, R2, I1, or I2. If `UMItags` is `FALSE`, then only R1 and R2 file types are required for analysis, if `UMItags` is `TRUE`, then I2 is a required file type as well.

barcode{1/2}Length
Integer values indicating the number of nucleotides in the barcodes or indexing sequences.

barcode{1/2}
Character values (i.e. `"I1"`) indicating which reads to find the associated indexing information for demultiplexing.

bc{1/2}Mismatch
An integer value indicating the number of tolarated mismatches in the barcode sequences for either barcode 1 or 2.

Sequence trimming

R{1/2}leadMismatch
Integer values indicating the number of allowed mismatches in either R1 or R2 leading sequence trimming. Recommend to set to less than 10% error.

R2odnMismatch
Integer value indicating the number of allowed mismatches in the unprimed ODN sequence, typically should be set to 0.

R{1/2}overMismatch
Integer values indicating the number of allowed mismatches in either R1 or R2 overreading trimming. This is converted into a percent matching and should be thought of as a number of mismatches allowed out of the total length of the overreading trim sequence.

R{1/2}overMaxLength
Searching for overread trimming in sequences can be time consuming while not producing different results. For this the total length of searched for sequences can be limited here. For example, if `ATGCGTCGATCGTACTGCGTTCGAC` is used as the overreading sequence, and 5 mismatches are allowed, then the tolerance will be 5/25 or 80% matching, but only the first 20 nucleotides of the sequence will be aligned for overtrimming, `ATGCGTCGATCGTACTGCGT`. With an 80% matching requirement, 16 out of 20 nucleotides will need to align for overread trimming to be initiated.

Binning

bins
A number of bins to separate filtered sequences into for higher parallel processing. The increasing the number of bins can help spread out the work required for processing to keep memory requirements lower.

level
A number indicating the number of reads that should be targeted for each bin. Bins will be filled to the level amount, leaving remaining bins empty if previous bins contain all the reads. Additionally, if all bins will "overflow", then reads will be evenly distributed across the number of bins.

Reference Alignment

BLATparams
A character string to be included with the BLAT call. A suggested example has been provided in the simulation config file. For options, please see the BLAT help options by typing `blat` into the commandline after activating `iguide`.

BWAparams
A character string to be inclued with the BWA call. A suggested example has been provided in the simulation config file. For options, please see BWA help by typing `bwa mem` into the commandline after activating `iguide`.

Post-alignment filtering

maxAlignStart
Integer value indicating the number of nucleotides at the beginning of the alignment that will be allowed to not align. Another way of thinking of this is the maximum start position on the query rather than the target reference. A default value of 5 means that the alignment needs to start in the first 5 nucleotides or the alignment is discarded during quality control filtering.

minPercentIdentity
This is a value between 0 and 100 indicating the minimum global percent identity allow for an alignment. If an alignment has less, then it is discarded during quality control filtering.

{min/max}TempLength
Specify the minimum (min) and maximum (max) template length expected. Joined alignments between R1 and R2 the are outside of this range are considered artifacts and are discarded or classified as chimeras.

Post-processing

refGenes / oncoGeneList / specialGeneList
These are special reference files in either text or BioConductoR's GenomicRanges objects. They can be in an '.rds' format or table format ('.csv' or '.tsv'). The `file` parameter should indicate the file path to the file (relative paths should be relative to the SnakeFile), and the `symbolCol` parameter should indicate the column in the data object which contains the reference names to be used in the analysis.

maxTargetMismatch
The maximum number of mismatches between the reference genome and target sequence allowed for consideration to be a target matched incorporation site. This is an integer value and is compared to the target sequence(s).

upstreamDist
The distance upstream of the incorporation site to look for a target similar sequence within the criteria specified by `maxTargetMismatch`.

downstreamDist
The distance downstream of the incorporation site to look / include for a target similar sequence within the criteria specified by `maxTargetMismatch`.

pileUpMin
An integer value indicating the number of alignments required to overlap before being considered a 'pileUp'.

recoverMultihits
While multihit alignments are often difficult to analyze, some information can still be gleamed from the data given reasonable assumptions. Adjusting this parameter to `TRUE` will still only focuses on sites that are uniquely mapped, but if a multihit includes a unique site and other locations, contributions are given to the unique site location. Further, reads and their contributions, umitags and fragments, are not double counted but instead evenly distributed to all included unique sites. **Note**, some sequencing artifacts may arrise in "off-target" associated sites. Users should be careful to conclude anything from these alignment artifacts. Leaving this option as `FALSE` is recommended if the user does not have a target sequence that locates a repetitive sequence.

Report

suppFile
Logical (`TRUE` or `FALSE`), if the supplemental file provided in `Supplemental_Info` should be used in the default report generated at the end of processing. If set to `FALSE`, the `Supplemental_Info` parameter is not required for processing.

{tables/figures}
: Logicals indicating if tables and figures should be generated from the report. Data will be included under the `reports` directory in the project run directory. For figures, both PDF and PNG formats will be generated if set to `TRUE` at 300 dpi while tables will be generated in a comma-separated values (csv) format.

reportData
: Logical indicating if a RData object should be saved during the report generation in the `reports` directory.

infoGraphic
: Logical indicating if an info graphic displaying the genomic distribution of incorporations should be generated at the beginning of the report. While aesthetically pleasing, the graphic gives the report a unique twist and can provide the knowledgeable user with information about the report at the very beginning.

signature
: Character string included at the beginning of reports to denote the author, analyst, laboratory, etc. Make sure you change if you don't want Chris getting credit for your work.

2.6.4 Nuclease Profiles

An additional component to the first part of the config file, is the Nuclease Profiles. The user can specify which nuclease they are using and include and profile to help identify edit sites. Nuclease can range from Cas9 to Cpf1 or TALEN based nickases.

Note: For TALEN and dual flanking nickases or nucleases, each side will need to be input as a different target. Specify in `Target_Sequences` the sequence and `On_Target_Sites` the actual editing site. Make sure you include two distinct identifiers for the sequences on-target sites, then specify the target treatment as `{target_seq1};{target_seq2}`.

Any name can be given in the `Nuclease` section, but that name needs to match the profile name as well. So if you want to call it "Cas9v2", then just make sure you have a profile named "Cas9v2".

Below is some ascii art that indicates the differences between nucleases. Additionally, below the art are example profiles for input into the iGUIDE software.:

```
Editing strategies by designer nucleases
Cas9 :
                  ><   PAM
ATGCATGCATGCATGCATGCA TGG (sense strand)

 TGCATGCATGCATGCATGCA NGG # gRNA
 |||||||||||||||||||| |||
TACGTACGTACGTACGTACGT ACC (anti-sense strand)
                  ><       # Dominant cutpoint

Cpf1 : Also known as Cas12a (similar nuclease structure for CasX)
                        ><         # Dominant cutpoint
GTTTG ATGCATGCATGCATGCATGCATGCATGC (sense strand)
  PAM
 TTTV ATGCATGCATGCATGCATGCA        # gRNA, nuclease activity leave overhang
 |||| |||||||||||||||||||||
CTAAC TACGTACGTACGTACGTACGTACGTACG (anti-sense strand)
                             ><     # Dominant cutpoint

TALEN : Protin-DNA binding domain fused with FokI nickase
ATATATATATATATATATAT GCATGCATGCATGCAT GCGCGCGCGCGCGCGCGCGC (sense strand)
\\\\\\\\\\\\\\\\\\\\
                     |------->
```

(continues on next page)

(continued from previous page)

```
                              <-------|
                                       \\\\\\\\\\\\\\\\\\\
TATATATATATATATATATA CGTACGTACGTACGTA CGCGCGCGCGCGCGCGCGCG (anti-sense strand)
# Proteins bind flanking the cleavage site and cut in the "insert" sequence.

CasCLOVER : Clo051 or another nickases with CRISPR-based binding domains
ATCCT ATGCATGCATGCATGCATGC TTAACCGGTTAACCGG TACGTACGTACGTACGTACG CGGTC
  ||| |||||||||||||||||||                                  (sense strand)
  PAM    Target Sequence  \------->
                                    <-------\   Target Sequence   PAM
(anti-sense strand)                          ||||||||||||||||||||| |||
TAGGA TACGTACGTACGTACGTACG AATTGGCCAATTGGCC ATGCATGCATGCATGCATGC GCCAG
```

Below are the example profiles.:

```
Nuclease_Profiles :
  Cas9 :
    PAM : "NGG"
    PAM_Loc : "3p"
    PAM_Tol : 1
    Cut_Offset : -4
    Insert_size : FALSE

  Cpf1 :
    PAM : "TTTV"
    PAM_Loc : "5p"
    PAM_Tol : 1
    Cut_Offset : 26     #(Anywhere between 23 and 28)
    Insert_size : FALSE

  CasX :
    PAM : "TTCN"
    PAM_Loc : "5p"
    PAM_Tol : 1
    Cut_Offset : 22     #(Anywhere between 16 and 29)
    Insert_size : FALSE

  TALEN :
    PAM : FALSE
    PAM_Loc : FALSE
    PAM_Tol : 0
    Cut_Offset : Mid_insert
    Insert_size : "15:21"

  CasCLOVER :
    PAM : "NGG"
    PAM_Loc : "3p"
    PAM_Tol : 1
    Cut_Offset : Mid_insert
    Insert_size : "10:30"
```

Profile parameters

PAM
protospacer adjacent motif - should be specified here and can contain ambiguous nucleotides.

PAM_Loc
indicates the location of the PAM with respect to the pattern, either '5p', '3p' or FALSE.

PAM_Tol
indicates the tolerance for mismatches in the PAM sequence (ignorned if PAM is FALSE).

Cut_Offset
indicates the offset from the 5' nucleotide of the PAM sequence where the nuclease creates a double strand break, unless PAM is FALSE, then the 5' position of the target sequence (also accepts "mid_insert" to specify middle of region between paired alignments).

Insert_size
is used if target sequences are expected to flank each other for editing, such as with TALENs, and indicates the expected size of the insert. To input a range, delimit the min and max by a colon, ie. 15:21. All names of nucleases used to treat specimens need to have a profile. Additional profiles should be added under the 'Nuclease_Profiles' parameter.

2.7 Sample Information Files

Sample information files (or sampleInfo files) contain information that may change from specimen to specimen. They need to contain at lease 3 columns of information: sample names, barcode 1, and barcode 2 sequences. Additionally, other parameters defined in the config file can be defined in the sample information file if they change from specimen to specimen.

Run specific config file will need to point to the sample information files. For convienence, a directory can be found at `iGUIDE/sampleInfo/` for depositing these files.

SampleInfo files need to have a specific naming format that follows '{RunName}.sampleinfo.csv'.

An appropriate format for the sample names is "{specimen}-{rep}" where 'specimen' is an alpha-numeric designator for the specimen and 'rep' is a numeric identifier for technical or biological replicates, separated by a dash (-). Replicates will be pooled during the final analysis, so if you want them to be separate in the report, make sure you give each specimen a different identifier.

For example, iGSP0002-1 and iGSP0002-2, will be pooled together for the report and analysis, but iGSP0002-1 and iGSP0003-1 will not. These names will be used in naming files, so do not include any special characters that will confuse file managment. Try to stick to common delimiters, such as - and _. Using a dot, ., as a delimiter is not currently supported.

A good practice is to put specimen identifiers at the beginning, replicate identifiers at the end following a "-", and anything else descriptive in the middle. For example, iGSP0002-neg-1, can specify the orientation the sample was processed with.

2.8 Supplemental Information Files

Supplemental information files (or supp files) contain information that may change from specimen to specimen. They have only one required column, "Specimen", but subsequence columns will be used to define conditions. Let's use the below supp file as an example.:

```
# Supplemental csv file example, padding included for visualization
Specimen, Nuclease, gRNA
iGXA,     Cas9,     TRAC
```

(continues on next page)

(continued from previous page)

```
iGXB,     Cas9,     TRAC
iGXC,     Cas9,     B2M
iGXD,     Cas9,     B2M
iGXE,     Mock,     Mock
iGXF,     Mock,     Mock
```

This type of setup would indicate that there are 6 specimens to be analyzed (iGXA - iGXF). Each of these would correlate with their sampleName'd replicates, so for iGXA, all samples with the format iGXA-{number} or iGXA-{info}-{number} would be pooled into the iGXA specimen.

Additionally, there are three conditions, defined by the distinct data excluding information in the "Specimen" column. So in this case, the conditions are "Cas9-TRAC", "Cas9-B2M", and "Mock-Mock". Within the report format, there are several analyses that are conditionally based rather than specimen based. This adds to the flexibility and utility of the reporting functions supplied with iGUIDE.

If the user would rather ever specimen analyzed independently and reported in that manner, then they can either run a report without a supp file or in a supp file include a column that distinguishes each specimen from each other.

Column names and formating are transferred directly into the report. Additionally, this files sets the order presented in the report. If "iGXC" comes before "iGXB" in the supp file, the it will be orderd as so throughout the report. Conditions, as well, follow this format. As presented above, the report will order the conditions in the following order "Cas9-TRAC", "Cas9-B2M", and "Mock-Mock", which is the order of first observation.

2.9 Setup a Run

Once the config and sampleInfo files have been configured, a run directory can be created using the command below where {ConfigFile} is the path to your configuration file:

```
cd path/to/iGUIDE
iguide setup {ConfigFile}
```

The directory should look like this (RunName is specified in the ConfigFile):

```
> tree analysis/{RunName}
analysis/{RunName}/
├── config.yml -> {path to ConfigFile}
├── input_data
├── logs
├── output
├── process_data
└── reports
```

Components within the run directory:

- config.yml - This is a symbolic link to the config file for the run
- input_data - Directory where input fastq.gz files can be deposited
- logs - Directory containing log files from processing steps
- output - Directory containing output data from the analysis
- process_data - Directory containing intermediate processing files
- reports - Directory containing output reports and figures

As a current convention, all processing is done within the analysis directory. The above command will create a file directory under the analysis directory for the run specified in by the config ('/iGUIDE/analysis/{RunName}'). At the end of this process, iGUIDE will give the user a note to deposit the input sequence files into the /analysis/{RunName}/input_data directory. Copy the fastq.gz files from the sequencing instrument into this directory if you do not have paths to the files specified in the config file.

iGUIDE typically uses each of the sequencing files (R1, R2, I1, and I2) for processing since it is based on a dual barcoding scheme. If I1 and I2 are concatenated into the read names of R1 and R2, it is recommended the you run `bcl2fastq ... --create-fastq-for-index-reads` on the machine output directory to generate the I1 and I2 files.

As iGUIDE has its own demultiplexing, it is recommend to not use the Illumina machine demultiplexing through input of index sequences in the SampleSheet.csv. If your sequence data has already been demultiplexed though, please see the *User Guide* for setup instructions.

2.10 List Samples in a Run

As long as the config and sampleInfo files are present and in their respective locations, you can get a quick view of what samples are related to the project. Using the `iguide list_samples` command will produce an overview table on the console or write the table to a file (specified by the output option). Additionally, if a supplemental information file is associated with the run, the data will be combined with the listed table.:

```
> iguide list_samples configs/simulation.config.yml

Specimen Info for : simulation.

 specimen   replicates       gRNA        nuclease
---------- ------------ --------------- ----------
   iGXA         1            TRAC         Cas9v1
   iGXB         1        TRAC;TRBC;B2M    Cas9v1
   iGXD         1             NA            NA
```

2.11 Processing a Run

Once the input_data directory has the required sequencing files, the run can be processed using the following command:

```
cd path/to/iGUIDE/
iguide run {ConfigFile}
```

Snakemake offers a great number of resources for managing the processing through the pipeline. I recommend familiarizing yourself with the utility (https://snakemake.readthedocs.io/en/stable/). Here are some helpful snakemake options that can be passed to iGUIDE by appending to the iguide command after `--`:

- `[--cores X]` multicored processing, specified cores to use by X.
- `[--nolock]` prevents locking of the working directory, allows for multiple sessions to run at the same time.
- `[--notemp]` keep all temporary files which are otherwise removed.
- `[-k, --keep-going]` will keep processing if one or more job error out.
- `[-w X, --latency-wait X]` wait X seconds for the output files to appear before erroring out.
- `[--restart-times X]` X is the number of time to restart a job if it fails. Defaults to 0, but is used in `iguide` to increase memory allocation.

- [--resources mem_mb=X] Defined resources, for iguide the mem_mb is the MB units to allow for memory allocation to the whole run. For HPC, this can be coupled with --cluster-config to request specific resources for each job.
- [--rerun-incomplete, --ri] Re-run all jobs that the output is recognized as incomplete, useful if your run gets terminated before finishing.
- [--cluster-config FILE] A JSON or YAML file that defines wildcards used for HPC.

2.12 Outputs and Reports

After the iguide run command has completed, the final run directory will contain a number of output and report files depending on the config parameters. Additionally, the if user is content with the analysis, they can use the iguide clean command to "clean up" the run directory. This will remove input data files, log files, and any remaining process data files, but will leave output and report files. This makes the "cleaned" run directories still compatible with the auxiliary workflow. A clean run directory will look something like the below tree.:

```
> tree analysis/{RunName}
analysis/{RunName}/
├── config.yml -> {path to ConfigFile}
├── input_data
├── process_data
├── logs
├── output
| ├── incorp_sites.{RunName}.rds
| ├── stats.core.{RunName}.csv
| └── stats.eval.{RunName}.csv
└── reports
  ├── report.{RunName}.html
  ├── runstats.{RunName}.html
  └── summary.{RunName}.txt
```

There are several standard output files. The incorp_sites.{RunName}.rds is the intermediate object that can be reprocessed into final data object and reports if the user would like to change most parameters. The stats files contain processing related information in a condensed form. These stats can be viewed in a more interpretable fashion from the runstats.{RunName}.html report.

The report.{RunName}.html would be the main data analysis report. The summary is a similar report but in a text based format. These are ample descriptions within the report template that will be included with the report. But if the user would like to customize this report, then they can modify the report template, found tools/rscripts/report_templates/iGUIDE_report_template.Rmd. Custom Rmd templates can also be provided through the iguide report command which will use eval output objects to "knit" reports in html or pdf output formats.

2.13 Contacts

Should you have any questions or comments and would like to contact the maintainer and designer of the iGUIDE software, please send a email to Chris [dot] L [dot] Nobles [at] Gmail [dot] com, with iGUIDE in the subject.

CHAPTER

THREE

CHANGELOG

v1.1.2 (April 17th, 2025)

- Resolved a bug dealing with factor levels during auxiliary workflow solutions.
- Added a second simulation data set (B) and labeled the original simulation data set (A).
- Expanded tests to cover auxiliary workflow solutions.

v1.1.1 (December 16th, 2024)

- Added reference gene lists to *./genomes* directory, as well as updated versions.
- Resolved bug associated with recovering multihit sites during analysis.
- Added option for Anaconda testing in test script to support custom installs. Try: *bash etc/tests/test.sh iguide 1 anaconda* with an anaconda install.
- Added functionality for more compatible gene lists between reference gene sets used for enrichment analysis.
- Updated sections of the documentation.

v1.1.0 (March 8th, 2020)

- Modified how samples designated as Mock are treated during the analysis
- Mock samples can now be indicated by "None" or "Control" as well (case-insensitive)
- Abundance can now be selected as [Read], [UMI], or [Fragment]{default} within config parameters and this selection will identify the abundance method used for analysis
- Added support for alternative UMI method (dx.doi.org/10.17504/protocols.io.wikfccw)

v1.0.2 (February 15th, 2020)

- Bugfix: UMItags set to FALSE will now process through to completion
- Rebuild: Updated to build version 1.0.1

v1.0.1 (December 3rd, 2019)

- Bugfix: Updated Gene set enrichment test within report

v1.0.0 (August 15th, 2019)

- Complete support for BLAT and BWA aligners
- Included a binning system to distribute workload into smaller loads
- Implemented a version tracking system into the intermediate data files (incorp_sites)
- Updated CLI with "hints" for snakemake processing

v0.9.9 (August 9th, 2019) - Additional updates

- Implemented support for BWA aligner
- Added tools (samqc) for working with other SAM/BAM output aligners as well
- Switched iguide support code to iguideSupport R-package and added unit tests
- Fixed bugs related to quoted table inputs (csv/tsv)
- Implemented a method to skip demultiplexing, see documentation for setup
- Resoved a number of issues identified, check GitHub for history!

v0.9.9 (June 10th, 2019)

- Revised the iGUIDE Report format to be more informational and clearer
- Revised a bit of the workflow to make reprocessing smoother
- Updated BLAT coupling script to be more memory efficient
- Fixed TravisCI testing!
- Changed stat workflow, now restarting analysis won't initiate a total reprocessing.
- Modified the assimilate + evaluate workflow
- Assimilate now only includes reference genome data, meaning a cleaner intermediate file
- Evaluate will now handle ref. gene sets and further analysis
- This increases the modularity and consistancy of the workflow

v0.9.8 (April 19th, 2019)

- iGUIDE can now support non-Cas9 nucleases as well!
- Implemented nuclease profiles into configs
- Updated assimilation, evaluation, and reporting scripts
- Added default resources to allow simpler HPC processing
- Included flexible system for identifying on-target sites
- Config can accept a range rather than a single site
- Acceptable notation: chr4:+:397-416 and chr3:*:397
- Changed build nomenclature from v0.9.3 to b0.9.3, so as not to confuse with version
- Added 'summary' subcommand to generate a consise text-based report
- Added short stats-based report to be produced at the end of processing
- Additional bugfixes.

v0.9.7 (March 6th, 2019)

- Hotfix to workflow.
- Changed 'setup' subcommand to python script based rather than snakemake.
- Changed file organization.

v0.9.6 (March 5th, 2019)

- Introduced process workflow steps: assimilate and evaluate
- Assimilate aligned data and compare with targeting sequences
- Incorp_sites now a core data object that can be combined across runs

- Evaluated data incorporates reference data and statistical models
- A staple data object for reports and can be constructed from multiple runs
- Included new subcommands 'eval' and modified 'report', report from either config(s) or eval dataset
- Cleaned up file structure
- Updated documentation in code and docs.
- Implemented accuracy and retention checks with simulation dataset.
- Updated simulation dataset with larger set to test analysis.

v0.9.5 (February 19th, 2019)

- Updated demultiplexing to be more efficient and better HPC compatible.
- Added RefSeq Extended reference gene sets
- 'ext' includes curated, predicted, and other RefSeq sets
- 'ext.nomodel' includes only curated and other RefSeq sets
- Incorporated resource allocation for job dependent memory consumption, works great with HPC to specify memory requirements
- Streamlined input for report generation by only requiring config(s)

v0.9.4 (January 30th, 2019)

- Updated 'report' utility and formating. Custom templates now accepted. Included as subcommand, check with 'iguide report -h'. PDF and HTML options report 'nicely' even when printed from either
- Updated build to v0.9.2 to support new formating in report
- Builds are constructed from spec files rather than yaml requirements
- Included the 'clean' subcommand to reduce size of processed projects. After cleaning a project, only terminal data files will remain

v0.9.3 (January 11th, 2019)

- Added 'list_samples' subcommand to list samples within a project.
- Caught a few bugs and worked them out for smoother processing and reports.

v0.9.2 (January 7th, 2019)

- Modified test dataset to run tests quicker and implemented CirclCI checking.

v0.9.1 (January 6th, 2019)

- Fixed problematic install for first time conda installers.

v0.9.0 (January 4th, 2019)

- Initial release.
- Supports setup and analysis of GUIDE-seq and iGUIDE experiments.
- Documentation on [ReadTheDocs.io](https://iguide.readthedocs.io/en/latest/index.html).