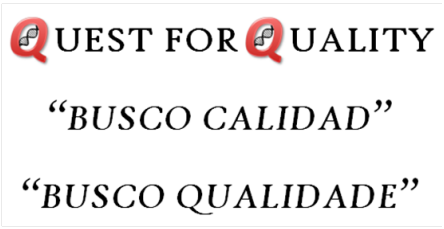


http://busco.ezlab.org

Version 3.0.2; July 2017

This document was last updated: 14 July 2017

BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs

Felipe A. Simão, Robert M. Waterhouse, Panagiotis Ioannidis,
Evgenia V. Kriventseva, & Evgeny M. Zdobnov

with BUSCO v3 contributions from Mathieu Seppey, Mosè Manni, & Fredrik Tegenfeldt

Bioinformatics (2015) 31 (19): 3210-3212
Abstract DOI: 10.1093/bioinformatics/btv351 PMID: 26059717
First published online: June 9, 2015. Article Metrics
Citations: Google Scholar, PubMed

Zdobnov's Computational Evolutionary Genomics Group: http://cegg.unige.ch

Department of Genetic Medicine and Development, University of Geneva Medical School
and Swiss Institute of Bioinformatics, rue Michel-Servet 1, 1211 Geneva, Switzerland.

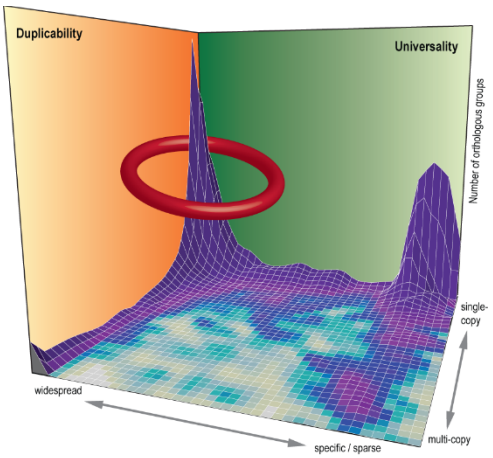
Please send questions (after reading this user guide) to: support@orthodb.org

Contents

Introduction

BUSCO completeness assessments employ sets of Benchmarking Universal Single-Copy Orthologs from OrthoDB (www.orthodb.org) to provide quantitative measures of the completeness of genome assemblies, annotated gene sets, and transcriptomes in terms of expected gene content. Genes that make up the BUSCO sets for each major lineage are selected from orthologous groups with genes present as single-copy orthologs in at least 90% of the species. While allowing for rare gene duplications or losses, this establishes an evolutionarily-informed expectation that these genes should be found as single-copy orthologs in any newly-sequenced genome. The evolutionary expectation means that if the BUSCOs cannot be identified in a genome assembly or annotated gene set, it is possible that the sequencing and/or assembly and/or annotation approaches have failed to capture the complete expected gene content.

An evolutionary expectation of gene content.



Classifying orthologs according to their universality (from widespread to specific or sparse species presence) and their duplicability (from mostly multi-copy to mostly single-copy) reveals an orthology landscape. BUSCOs are selected from orthologous groups with single-copy orthologs in the majority of species (circled in red). Thus, BUSCO searches are expected to find matching single-copy orthologs in any newly-sequenced genome from the appropriate species clade. Figure adapted from the *Drosophila melanogaster* insect orthology landscape in Waterhouse, Current Opinion in Insect Science, 2015.

BUSCO sets were first defined using orthologs from OrthoDB v7 as described in Waterhouse *et al*. Nucleic Acids Research, 2013, PMID: 23180791, and were subsequently incorporated into the BUSCO assessment tool as described in Simão *et al*. Bioinformatics, 2015, PMID: 26059717. BUSCO v2 implemented improvements to the underlying analysis software as well as updated and extended sets of BUSCOs covering additional lineages based on orthologs from OrthoDB v9 (Zdobnov *et al*. Nucleic Acids Research, 2017, PMID: 27899580). BUSCO v3 maintains all v2 features and uses the same lineage datasets. It is the result of a major refactoring of the codebase, converting the original single BUSCO script into a python package available system-wide, v3 now also makes use of a user-editable configuration file in addition to the standard command line arguments. The assessment tool implements a computational pipeline to identify and classify BUSCO group matches from genome assemblies, annotated gene sets, or transcriptomes, using HMMER hidden Markov models and *de novo* gene prediction with Augustus. Running the assessment tool requires working installations of Python, HMMER, Blast+, and Augustus (genome assessment only). Genome assembly assessment first identifies candidate regions to be assessed with tBLASTn searches using BUSCO consensus sequences. Gene structures are then predicted using Augustus with BUSCO block profiles. These predicted genes, or all genes from an annotated gene set or transcriptome, are then assessed using HMMER and lineage-specific BUSCO profiles to classify matches. The recovered matches are classified as 'complete' if their lengths are within the expectation of the BUSCO profile match lengths. If these are found more than once they are classified as 'duplicated'. The matches that are only partially recovered are classified as 'fragmented', and BUSCO groups for which there are no matches that pass the tests of orthology are classified as 'missing'.

Software setup

Installation

[0] BUSCO has been developed and tested on Linux (e.g. Arch Linux, CentOS, Ubuntu) and can be run on MacOS X, provided you are able to install Augustus properly, as issues have been reported. We recommend using a Linux box for running BUSCO with its installed dependencies, and cannot provide support for MacOS and Windows operating systems. As an alternative to setting up BUSCO on your own machine, you can use the BUSCO virtual machine (see next section for details).

[1] The BUSCO assessment software distribution is available from the public **GitLab** project: https://gitlab.com/ezlab/busco where it can be downloaded or cloned using a git client (git clone https://gitlab.com/ezlab/busco.git). We encourage users to opt for the git client option in order to facilitate future updates.

[2] BUSCO is written for Python 3.x and Python 2.7+. It runs with the standard packages. We recommend using Python3 when available.

[3] BUSCO v3 requires its packages to be installed on the system by running **`setup.py`** on the version of python chosen to run the tool. You can run it with root privileges or for the current user only by choosing one of these two possibilities:

```
sudo python setup.py install
```

```
python setup.py install --user
```

Note: You need the BUSCO main folder to be your current directory when running **`setup.py`**.

[4] BUSCO v3 employs a user-editable configuration file for defining required settings and parameters (previously set through the command line arguments). In the **`config/`** subfolder the **`config.ini.default`** file must first be copied to **`config.ini`** and then edited before running BUSCO. In this file, you must declare the paths to all dependencies (see below) and you can optionally define the required input parameters (described later in this document). Note: providing input parameters through the command line will override those defined in **`config.ini`**. The **`config.ini.default`** file is extensively commented and self explanatory.

Additionally, you can define a custom path (including the filename) to the **`config.ini`** file by setting the following environment variable, which will override the default location:

```
export BUSCO_CONFIG_FILE="/path/to/filename.ini"
```

This is useful for switching between configurations or in a multi-users environment.

[5] In addition to **Python,** you will need to make sure that the following required software packages are installed with their paths declared in the **`config.ini`** file.

- **NCBI BLAST+** **[NB: please see release note 2.0.1 below]**
  https://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST

- **HMMER (HMMER v3.1b2)**
  http://hmmer.org/

- **Augustus (> 3.2.1) (**only used for assessing genomes)
  http://bioinf.uni-greifswald.de/augustus/
  Augustus uses several executables and PERL scripts. Please refer to Augustus documentation for PERL requirements.
  In addition to the entries in the **`config.ini`** file, Augustus requires environment variables to be declared as follows:

```
export PATH="/path/to/AUGUSTUS/augustus-3.2.3/bin:$PATH"
export PATH="/path/to/AUGUSTUS/augustus-3.2.3/scripts:$PATH"
export AUGUSTUS_CONFIG_PATH="/path/to/AUGUSTUS/augustus-3.2.3/config/"
```

NB: you can use the **`printenv`** command to view all your environment settings.

Please make sure that each of the three software packages listed above work **INDEPENDENTLY** of BUSCO before attempting to run any BUSCO assessments.

How to solve: **'`ERROR Cannot write to Augustus config path` …'**

If Augustus is installed globally on your system and you do not have administrator rights there is a simple workaround that should work on most systems. This is because during genome mode assessments Augustus needs to write gene model prediction parameters to its own '**`config`**' directory, and if you do not have write access to this directory the analysis will fail. Simply recursively copy the entire Augustus '**`config`**' directory to a location where you do have write access, and then set the **`AUGUSTUS_CONFIG_PATH`** variable to this location.

```
cp -r /path/to/AUGUSTUS/augustus-3.2.3/config /my/home/augustus/config
export AUGUSTUS_CONFIG_PATH="/my/home/augustus/config/"
```

[6] Depending on the species you wish to assess, you should now download the appropriate lineage-specific profile libraries and accompanying information from http://busco.ezlab.org to your **`BUSCO`** directory: for example, *actinopterygii_odb9, arthropoda_odb9, ascomycetes_odb9, aves_odb9, bacteria_odb9, diptera_odb9, endopterygota_odb9, eukaryota_odb9, fungi_odb9, hymenoptera_odb9, insecta_odb9, mammalia_odb9, metazoa_odb9, or vertebrata_odb9.*

Virtual machine

The BUSCO assessment tool and its dependencies (e.g. BLAST, HMMER, Augustus) have been set up on a virtual machine (VM) that can be downloaded from http://busco.ezlab.org/. The **Ubuntu GNOME 32-bit BUSCO VM** was built using OSboxes (http://www.osboxes.org/) and can be launched with VM software such as VMware (https://www.vmware.com/) or VirtualBox (https://www.virtualbox.org/), so you will need to download and install the most appropriate version (e.g. for Windows, Linux, Macintosh, or Solaris etc.) of the VM software for your system. Please note: **we cannot provide support for setting up the VM software**, please refer to their websites for the required setup information, and how to use them to launch a VM, especially if you want to configure it to be able to use multiple processors.

Once you have launched the BUSCO VM then you can run BUSCO as you would normally, e.g. by first creating a new working directory for a new project and arranging your input files accordingly. Simply right click and open a terminal, this will start you off in the `~/BUSCOVM/busco3` directory. Double click the '**Link to BUSCOVM**' icon to open the VM's directory explorer (contents detailed below). You will also need to download (from http://busco.ezlab.org/) and unpack the tarball of the lineage(s) that you intend to use: e.g. in the `lineages` directory, `tar -xf vertebrata_odb9.tar.gz`

The VM directories:

`augustus`	Contains Augustus software
`busco3`	Contains BUSCO software
`hmmer`	Contains HMMER software
`lineages`	Directory for BUSCO lineage datasets (download and unpack tarballs before use!)

Example:

From the `~/BUSCOVM/busco3` directory in the terminal, create a new working directory:

```
mkdir MyProject1
```

From that directory, get your assembly, transcriptome, or gene set that you wish to assess:

```
wget website/where_your/data_are_found/YOUR_SEQUENCE_FILE.fa
```

Then launch a BUSCO assessment of your data, e.g.

```
python ~/BUSCOVM/busco3/scripts/run_BUSCO.py
-i YOUR_SEQUENCE_FILE.fa -o OUTPUT_NAME
-l ~/BUSCOVM/lineages/NAME_OF_LINEAGE -m geno
```

Quick start BUSCO assessments

`-m` or `--mode` sets the assessment MODE: **`genome`**, **`proteins`**, **`transcriptome`**

1- Genome assembly assessment

```
python scripts/run_BUSCO.py      -i SEQUENCE_FILE -o OUTPUT_NAME
                                 -l LINEAGE -m geno
```

SEQUENCE_FILE	genome assembly file in FASTA format
OUTPUT_NAME	name to use for the run and all temporary files (appended)
LINEAGE	location of the BUSCO lineage data to use (e.g. eukaryota_odb9) (NB: without specifying a particular species, Augustus species parameters will be selected according to the predefined defaults)

2- Gene set (proteins) assessment

```
python scripts/run_BUSCO.py      -i SEQUENCE_FILE -o OUTPUT_NAME
                                 -l LINEAGE -m prot
```

SEQUENCE_FILE	gene set (protein amino acid sequences) file in FASTA format
OUTPUT_NAME	name to use for the run and temporary files (appended)
LINEAGE	location of the BUSCO lineage data to use (e.g. vertebrata_odb9)

3- Transcriptome assessment

```
python scripts/run_BUSCO.py      -i SEQUENCE_FILE -o OUTPUT_NAME
                                 -l LINEAGE -m tran
```

SEQUENCE_FILE	transcript set (DNA nucleotide sequences) file in FASTA format
OUTPUT_NAME	name to use for the run and temporary files (appended)
LINEAGE	location of the BUSCO lineage data to use (e.g. fungi_odb9)

Example assessment runtimes (gene sets with 5 CPUs, genomes with 12 CPUs):

Human genome (3.1 Gbp), assessed with 4'104 mammalian BUSCOs: 6 days 15 hours
Human gene set (20'398 proteins), assessed with 4'104 mammalian BUSCOs: ~20 minutes
Human genome (3.1 Gbp), assessed with 978 metazoan BUSCOs: ~21 hours
Human gene set (20'398 proteins), assessed with 978 metazoan BUSCOs: ~3 minutes
Drosophila genome (140 Mbp), assessed with 2'799 dipteran BUSCOs: ~1 hour 45 minutes
Drosophila gene set (13'954 proteins), assessed with 2'799 dipteran BUSCOs: ~14 minutes
Drosophila genome (140 Mbp), assessed with 978 metazoan BUSCOs: ~19 minutes
Drosophila gene set (13'954 proteins), assessed with 978 metazoan BUSCOs: ~2 minutes

NB: more fragmented genomes will take longer as second round searches and gene predictions are performed for BUSCOs found to be fragmented or missing after the first round.

Options

```
python scripts/run_BUSCO.py -i [SEQUENCE_FILE] -o [OUTPUT_NAME]
                            -l [LINEAGE] -m [MODE]
```

Any provided command line argument overrides its equivalent in the **config.ini** file.

A mandatory argument can only be omitted from the command line if it is defined in the **config.ini** file.

1- Mandatory arguments

-i SEQUENCE_FILE, --in SEQUENCE_FILE
Input sequence file in FASTA format (not compressed/zipped!).
Can be an assembled genome or transcriptome (DNA),
or protein sequences from an annotated gene set.
NB: select just one transcript/protein per gene for your input,
otherwise they will appear as 'Duplicated' matches.

-o OUTPUT_NAME, --out OUTPUT_NAME
Give your analysis run a recognisable short name.
Output folders and files will be labelled (prepended) with this name.
WARNING: do not provide a path.

-l LINEAGE, --lineage_path LINEAGE
Specify location of the BUSCO lineage data to be used.
Visit http://busco.ezlab.org/ for available lineages.

-m MODE, --mode MODE
Specify which BUSCO analysis mode to run.
There are three valid modes:
- geno or genome, for genome assemblies (DNA).
- tran or transcriptome, for transcriptome assemblies (DNA).
- prot or proteins, for annotated gene sets (protein).

2- Optional arguments

-c N, --cpu N
Specify the number (N=integer) of threads/cores to use (default: 1).

-e N, --evalue N
E-value cutoff for BLAST searches.
Allowed formats: 0.001 or 1e-03 (default: 1e-03).

-f, --force
Force rewriting of existing files/folders.
Must be used when output files with the provided name already exist.

-sp SPECIES, --species SPECIES
Name of existing Augustus species gene finding parameters.
See Augustus documentation for available options.
Each lineage has a default species (see below on assessment sets).
Selecting a closely-related species usually produces better results.

-t PATH, --tmp PATH
Where to store temporary files (default: ./tmp).

-z, --tarzip
Results folders with many files will be tarzipped.

-r, --restart
Restart the BUSCO run from the last successfully-completed step.
NB: If all the required results files from previous steps are not all found then this will not be possible.

--limit REGION_LIMIT
How many candidate regions to consider (integer, default: 3).
NB: this limit is on scaffolds, chromosomes, or transcripts, not individual hit regions.

--long
Turn on Augustus optimization mode for self-training (default: Off).
Adds substantially to the run time!
Can improve results for some non-model organisms.

-q, --quiet
Disable the info logs, display only errors.

--blast_single_core
Force tblastn to run on a single core and ignore the --cpu argument for this step only. Useful if inconsistencies when using multiple threads are noticed **[NB: please see release note 2.0.1 below]**

-v, --version
Show this version information and exit.

-h, --help
Show this help message and exit.

PASSING SPECIAL AUGUSTUS OPTIONS:

--augustus_options='--option1=value1 --option2=value2'
Allows for the passing of special options to Augustus.
E.g. **--augustus_options='--translation_table=6 -progress=true'**

Output

Successful execution of the BUSCO assessment pipeline in any mode will create a directory named **run_OUTPUT_NAME** where '**OUTPUT_NAME**' is your assigned name for the assessment run (set with the **-o OUTPUT_NAME** mandatory option). Similarly, individual results files from your assessment run will also be labelled with your run '**OUTPUT_NAME**'. This output results directory will contain several files and directories:

Where: **OUTPUT_NAME** in this example was set to **XXXX**

1- Main results files

short_summary_XXXX.txt
Contains a plain text summary of the results in BUSCO notation.
Also gives a brief breakdown of the metrics.

full_table_XXXX.tsv
Contains the complete results in a tabular format with scores and lengths of BUSCO matches, and coordinates (for genome mode) or gene/protein IDs (for transcriptome or proteins mode).

missing_buscos_list_XXXX.tsv
Contains a list of missing BUSCOs.

2- Results directories

hmmer_output
Tabular format HMMER output of searches with BUSCO HMMs.

translated_proteins
Transcript sequence translations, only created during transcriptome assessment.

blast_output
tBLASTn results, not created for assessment of proteins.
File: **tblastn_XXXX.txt** = tabular tBLASTn results
File: **coordinates_XXXX.txt** = locations of BUSCO matches (genome mode)

augustus_output
Augustus-predicted genes, only created during genome assessment.
File: **augustus.log** = full details on Augustus jobs
File: **training_set_XXXX.txt** = genes used for Augustus training
Folder: **predicted_genes** = Augustus raw gene output
Folder: **extracted_proteins** = Augustus protein FASTA output
Folder: **retraining_parameters** = Augustus species folder, created during BUSCO retraining. Specific to your species. Can be put back in Augustus species folder.
Folder: **gb** = GenBank format complete BUSCOs
Folder: **gffs** = General Feature Format complete BUSCOs

single_copy_busco_sequences (genome mode only)
FASTA format file for each complete single-copy BUSCO identified.
.faa files contain protein sequences
.fna files contain coding sequences.

Test with sample data

Sample data are provided to test your BUSCO setup. Execute the following commands and compare the final output '**run_TEST**' with the provided files in '**sample_data/run_SAMPLE**'.

1. Run BUSCO assessment on sequence file '**target.fa**' in genome mode using the '**example**' lineage, both found in the '**sample_data**' folder.

```
python scripts/run_BUSCO.py --in sample_data/target.fa --out TEST
--lineage_path sample_data/example --mode genome
```

or

```
python scripts/run_BUSCO.py -i sample_data/target.fa -o TEST -l
sample_data/example -m geno
```

2. Compare your final output '**run_TEST**' with the results provided files in '**run_SAMPLE**'.

Your results should be located in the folder '**run_TEST**':

Folder: **augustus_output**
Folder: **blast_output**
Folder: **hmmer_output**
Folder: **single_copy_busco_sequences**

File: **full_table_TEST.tsv**
File: **missing_buscos_list_TEST.tsv**
File: **short_summary_TEST.txt**

Example output: `short_summary_TEST.txt`

```
# BUSCO version is: 3.0.0
# The lineage dataset is: sample dataset BUSCO 2.0 (Creation date:
07.10.2016, number of species: 23, number of BUSCOs: 10)
# To reproduce this run: python scripts/run_BUSCO.py -i
sample_data/target.fa -o TEST -l sample_data/example -m genome -c 1 -f
#
# Summarized benchmarking in BUSCO notation for file
sample_data/target.fa
# BUSCO was run in mode: genome

      C:80.0%[S:80.0%,D:0.0%],F:0.0%,M:20.0%,n:10

      8     Complete BUSCOs (C)
      8     Complete and single-copy BUSCOs (S)
      0     Complete and duplicated BUSCOs (D)
      0     Fragmented BUSCOs (F)
      2     Missing BUSCOs (M)
      10    Total BUSCO groups searched
```

Example output: `full_table_SAMPLE.tsv`
Columns are: BUSCO ID, status, scaffold name, start, end, score, length

```
# BUSCO version is: 3.0.0
# The lineage dataset is: sample dataset BUSCO 2.0 (Creation date:
07.10.2016, number of species: 23, number of BUSCOs: 10)
# To reproduce this run: python scripts/run_BUSCO.py -i
sample_data/target.fa -o TEST -l sample_data/example -m genome -c 1 -f
#
BUSCO_1   Complete  sample    30184     31421     320.9     193
BUSCO_2   Complete  sample    31646     47625     872.0     443
BUSCO_3   Complete  sample    62761     68675     241.8     146
BUSCO_4   Complete  sample    69310     70561     420.3     418
BUSCO_5   Complete  sample    70677     71872     497.8     304
BUSCO_6   Complete  sample    81792     89171     1676.7    920
BUSCO_7   Complete  sample    168309    169293    423.6     243
BUSCO_8   Complete  sample    232880    234117    274.6     194
BUSCO_9   Missing
BUSCO_10  Missing
```

Assessment sets

1- BUSCO selections

As described briefly in the introduction, BUSCOs are selected from OrthoDB orthologous groups at major species radiations requiring orthologues to be present as single-copy genes in the vast majority (>90%) of species. BUSCO v1 sets were selected from OrthoDB v7, and BUSCO v2 sets were selected from OrthoDB v9. The selection of species for each lineage proceeds via a three-step process: (i) all orthologous groups with genes present in more than half the species for the given lineage are first identified for downstream processing (the >50% set); (ii) sets of closely-related species are then identified using the mean percent identity of all best reciprocal hits between each pair of species, and a representative for each set is chosen by identifying the species with the fewest number of missing orthologues in the >50% set of orthologous groups; (iii) these representatives (or singletons in the case of no close-relatives) are then assessed to sum the number of orthologous groups for which they contain a single-copy orthologue, multi-copy orthologues, or they are missing an orthologue. Those with significantly more multi-copy or missing orthologues are flagged for removal from the species set, but species holding key phylogenetic positions in the lineage (e.g. outgroup species, or species from subclades with few representatives) can be retained. Species that are removed (too closely-related or poorly-performing species) will not be used for building the final BUSCO profiles, but the set of removed species is still used as a filter as only orthologous groups where these species are >75% present and >50% single-copy will be retained. This step is iterated with manual selections at each stage until all species have either been selected to be retained or they do not have significantly more multi-copy or missing orthologous groups.

Additional filters may be added to ensure that outgroup species or species from subclades with few representatives are not missing disproportionately high numbers of orthologues. The mean percent identity used to define closely-related sets of species will vary according to the diversity of the set being assessed, the aim being to ensure that the species selection is not dominated by species from a few subclades. Selecting species to be retained (despite having significantly more multi-copy or missing orthologues) is a subjective process, guided by the species phylogeny and the resulting numbers of BUSCOs obtained.

These initial sets of BUSCOs are then used to build their corresponding BUSCO profiles. Those with low-quality protein sequence alignments, or that fail to produce BUSCO block profiles for use by Augustus are discarded. The remaining BUSCOs are then assessed for their accuracy at correctly recalling the orthologues that were used to build their BUSCO profiles (i.e. running BUSCO in protein mode on the full gene sets of the selected input species) and poorly-performing ones are discarded. Finally, a selection of genome assembly assessments are carried out and BUSCOs that consistently fail to recover complete gene predictions, despite these genes being present in the genome assemblies, are also discarded.

2- BUSCO lineages

The tables below provide detailed information on the numbers of species and orthologous groups selected for each BUSCO v2 lineage, and their default species for genome mode assessments (i.e. if the user does not specify a species from the parameter sets precomputed by Augustus).

For further information about the species sets, the orthologous groups, and their member genes/proteins, view the detailed information provided with each lineage download.

Each lineage dataset should contain:
Folder: **`hmms`** HMM file for each BUSCO
Folder: **`info`** Files with lists of species, genes, ortho-groups, and annotations
Folder: **`prfl`** Block profile file for each BUSCO
File: **`ancestral`** FASTA file, consensus ancestral sequences for each BUSCO
File: **`ancestral_variants`** FASTA file, consensus & variant sequences for each BUSCO
File: **`dataset.cfg`** Configuration data including default Augustus species
File: **`lengths_cutoff`** Length cut-offs for complete BUSCO matches
File: **`scores_cutoff`** Score cut-offs for orthologous BUSCO matches

Bacterial lineages:

Lineage	Number of species	Number of BUSCO groups	Augustus default species parameters
bacteria_odb9	3663	148	E_coli_K12
proteobacteria_odb9	1520	221	E_coli_K12
gammaproteobacteria_odb9	721	452	E_coli_K12
enterobacteriales_odb9	216	781	E_coli_K12
deltaepsilonsub_odb9	218	296	E_coli_K12
betaproteobacteria_odb9	215	582	E_coli_K12
rhizobiales_odb9	154	686	E_coli_K12
firmicutes_odb9	951	232	E_coli_K12
lactobacillales_odb9	330	443	E_coli_K12
clostridia_odb9	289	254	E_coli_K12
bacillales_odb9	238	526	s_aureus
actinobacteria_odb9	412	352	E_coli_K12
bacteroidetes_odb9	264	443	E_coli_K12
cyanobacteria_odb9	112	834	E_coli_K12
spirochaetes_odb9	102	237	E_coli_K12
tenericutes_odb9	68	166	E_coli_K12

Eukaryotic lineages:

Lineage	Number of species Total : Selected	Number of BUSCO groups	Augustus default species parameters
eukaryota_odb9*	90 : 65	303	fly
nematoda_odb9	10 : 8	982	caenorhabditis
metazoa_odb9*	330 : 65	978	fly
arthropoda_odb9	133 : 60	1'066	fly
insecta_odb9	116 : 42	1'658	fly
endopterygota_odb9	100 : 35	2'442	fly
diptera_odb9	53 : 25	2'799	fly
hymenoptera_odb9	32 : 25	4'415	honeybee1
vertebrata_odb9*	172 : 65	2'586	human
actinopterygii_odb9	23 : 20	4'584	zebrafish
tetrapoda_odb9	146 : 55	3'950	human
aves_odb9	54 : 40	4'915	chicken
mammalia_odb9	84 : 50	4'104	human
euarchontoglires_odb9	37 : 25	6'192	human
laurasiatheria_odb9	33 : 25	6'253	human
fungi_odb9	227: 85	290	aspergillus_nidulans
microsporidia_odb9	15 : 14	518	encephalitozoon_ cuniculi_GB
dikarya_odb9	208 : 75	1'312	aspergillus_nidulans
ascomycota_odb9	168 : 75	1'315	aspergillus_nidulans
saccharomyceta_odb9	161 : 70	1'759	saccharomyces_ cerevisiae_S288C
pezizomycotina_odb9	110 : 50	3'156	aspergillus_nidulans
sordariomyceta_odb9	50 : 30	3'725	fusarium_graminearum
eurotiomycetes_odb9	41 : 25	4'046	aspergillus_nidulans
saccharomycetales_odb9	51 : 30	1'711	saccharomyces_ cerevisiae_S288C
basidiomycota_odb9	40 : 25	1'335	coprinus
embryophyta_odb9*	31 : 20	1'440	arabidopsis
alveolata_stramenophiles_ensembl*	37 : 24	234	toxoplasma
protists_ensembl*	104 : 33	215	toxoplasma

* **eukaryota_odb9**: orthology delineation with 90 selected representative species.
* **metazoa_odb9**: excludes nematoda (despite being metazoans) because of high sequence divergence (for such worms use nematoda set instead).
* **vertebrata_odb9**: excludes the sea lamprey (*Petromyzon marinus*) because of high sequence divergence, should thus be used for Gnathostomata as it is not ideal for Agnatha.
* **embryophyta_odb9**: excludes the moss (*Physcomitrella patens*) and club-moss (*Selaginella moellendorffii*) because of high sequence divergence, should thus be used for Magnoliophyta (flowering plants).
* **alveolata_stramenophiles** and **protists**: from EnsemblGenomes Release 31

Plotting results

The **scripts/generate_plot.py** script allows users to quickly view their BUSCO summary results in an easily-understandable bar chart. The **scripts/generate_plot.py** uses **R** (https://www.r-project.org/) and **ggplot2** (http://ggplot2.org/) to summarize BUSCO runs for side-by-side comparisons. The script produces a PNG image (if both **R** and **ggplot2** are available), as well as an R source code file that can be used to run on a different machine where both **R** and **ggplot2** are available or which can be edited to fully customise the resulting bar chart (colours, labels, fonts, axes, etc.).

To run **scripts/generate_plot.py**, first create a folder, e.g. **mkdir BUSCO_summaries**, and then copy the BUSCO short summary file from each of the runs you want to plot into this folder.
e.g. **cp run_XX1/short_summary_XX1.txt BUSCO_summaries/.**
e.g. **cp run_XX2/short_summary_XX2.txt BUSCO_summaries/.**
e.g. **cp run_XX3/short_summary_XX3.txt BUSCO_summaries/.**

Then simply run the script giving as the only argument the name (or full path if you are not in same working directory) of the folder you created containing the summaries you wish to plot.
e.g. **python scripts/generate_plot.py -wd BUSCO_summaries**
e.g. **python scripts/generate_plot.py -wd /full/path/to/my/folder/BUSCO_summaries**

The resulting PNG image and the corresponding R source code file will be produced in the same folder containing the BUSCO summaries. By default, the run name is used as the label for each plotted result, and this is automatically extracted from the short summary file name: so for **short_summary_XX1.txt** the label would be **XX1**. You can modify this as long as you keep the naming convention: **short_summary_[edit_name_here].txt** or you can simply edit the R source code file to change any plotting parameters and produce a personalised bar chart running the code manually in your R environment.

python scripts/generate_plot.py -wd SUMMARIES_FOLDER

1 - Mandatory argument:

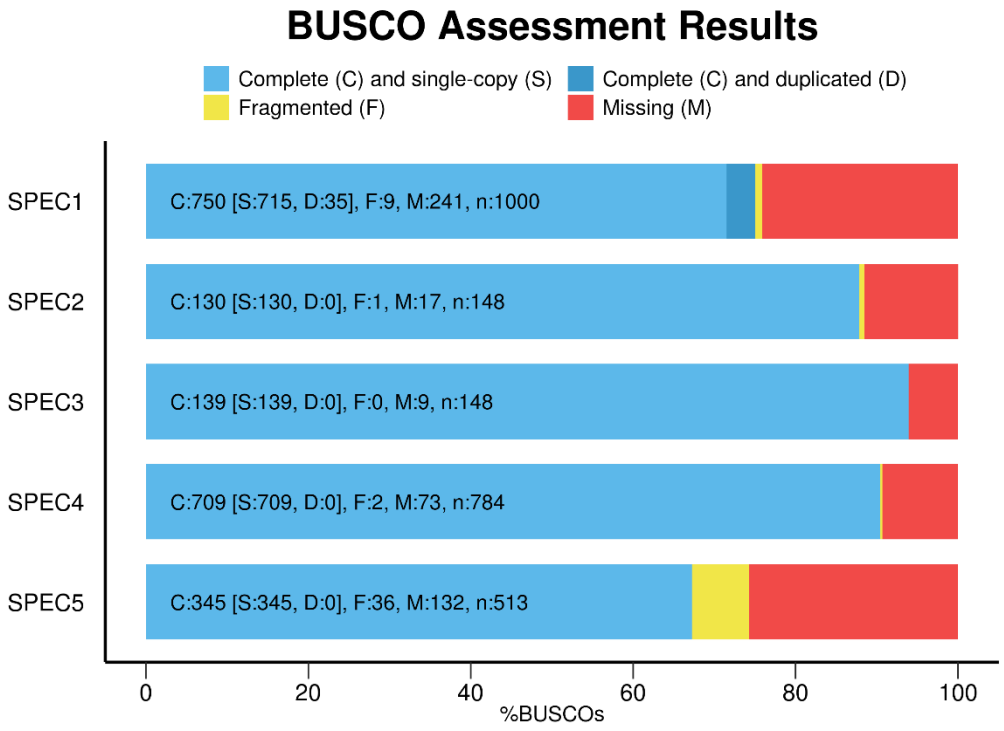
-wd PATH/NAME, --working_directory PATH/NAME
Name or full path to folder containing BUSCO short_summary files.

2 - Optional arguments:

--no_r	To avoid running R. It will just create the R script file in the working directory
-q, --quiet	Disable the info logs, displays only errors
-v, --version	Show this version and exit
-h, --help	Show this help message and exit

Example **scripts/generate_plot.py** bar chart:

```
mkdir my_summaries
cp run_SPEC1/short_summary_SPEC1.txt my_summaries/.
cp run_SPEC2/short_summary_SPEC2.txt my_summaries/.
cp run_SPEC3/short_summary_SPEC3.txt my_summaries/.
cp run_SPEC4/short_summary_SPEC4.txt my_summaries/.
cp run_SPEC5/short_summary_SPEC5.txt my_summaries/.
python scripts/generate_plot.py -wd my_summaries
```



Backward compatibility

BUSCO v2/v3 have been designed to be backward compatible with BUSCO v1 lineage datasets. However, warnings will be printed to the log files indicating missing information such as the dataset configuration files and the ancestral variant sequence files. BUSCO v2/v3 implement some changes to the way potential genomic regions are identified, the way scores are handled, and the way results are reported, thus the results using BUSCO v1 lineage datasets will not be the same as when running assessments with BUSCO v1 software.

Interpreting BUSCO results

BUSCO attempts to provide a quantitative assessment of the completeness in terms of expected gene content of a genome assembly, transcriptome, or annotated gene set. The results are simplified into categories of 'Complete and single-copy', 'Complete and duplicated', 'Fragmented', or 'Missing' BUSCOs. These labels are simplifications of the most likely scenario, described below along with other, less-likely but still theoretically possible, interpretations:

Complete
If found to be complete, whether single-copy or duplicated, the BUSCO matches have scored within the expected range of scores and within the expected range of length alignments to the BUSCO profile. If in fact an orthologue is not present in the input dataset, or the orthologue is only partially present (highly fragmented), and a high-identity full-length homologue is present, it is possible that this homologue could be mistakenly identified as the complete BUSCO. The score thresholds are optimised to minimise this possibility, but it can still occur.

Fragmented
If found to be fragmented, the BUSCO matches have scored within the range of scores but not within the range of length alignments to the BUSCO profile. For transcriptomes or annotated gene sets this indicates incomplete transcripts or gene models. For genome assemblies this could indicate either that the gene is only partially present or that the sequence search and gene prediction steps failed to produce a full-length gene model even though the full gene could indeed be present in the assembly. Matches that produce such fragmented results are given a 'second chance' with a second round of sequence searches and gene predictions with parameters trained on those BUSCOs that were found to be complete, but this can still fail to recover the whole gene. Some fragmented BUSCOs from genome assembly assessments could therefore be complete but are just too divergent or have very complex gene structures, making them very hard to locate and predict in full.

Missing
If found to be missing, there were either no significant matches at all, or the BUSCO matches scored below the range of scores for the BUSCO profile. For transcriptomes or annotated gene sets this indicates that these orthologues are indeed missing or the transcripts or gene models are so incomplete/fragmented that they could not even meet the criteria to be considered as fragmented. For genome assemblies this could indicate either that these orthologues are indeed missing, or that the sequence search step failed to identify any significant matches, or that the gene prediction step failed to produce even a partial gene model that might have been recognised as a fragmented BUSCO match. Like for fragments, BUSCOs missing after the first round are given a 'second chance' with a second round of sequence searches and gene predictions with parameters trained on those BUSCOs that are complete, but this can still fail to recover the gene. Some missing BUSCOs from genome assembly assessments could therefore be partially present, and even possibly (but unlikely) complete, but they are just too divergent or have very complex gene structures, making them very hard to locate and predict correctly or even partially.

Best practices

Some common sense advice on how to run BUSCO assessments, as well as on how to report BUSCO findings in publications etc. to make sure they are both interpretable and reproducible.

Running BUSCO:

- BUSCO has been tested with Augustus 3.2.1, 3.2.2, and 3.2.3 so we strongly recommend using one of these versions. Future updates to Augustus should not necessarily cause any serious problems, but users should be aware of this possibility, and that the BUSCO team will strive to test and update BUSCO when new versions of Augustus are released.
- Generally the lineage to select for your assessments should be the most specific lineage available, e.g. for assessing fish data one would select the '**`actinopterygii`**' lineage rather than the '**`metazoa`**' lineage.
- If you are assessing a large number of species/strains/versions etc. then to minimise runtime (at the expense of resolution) one might select a less specific lineage set with fewer BUSCOs, e.g. for assessing 20 bird genomes each with a couple of different assembly versions one might select the '**`vertebrata`**' or the '**`metazoa`**' lineages rather than the '**`aves`**' lineage, at least for the initial rounds of assessments.
- Assessments generally produce several folders with lots of files (especially when assessing genome assemblies). These are for your benefit, so that you can examine individual cases in more detail and/or use the data for downstream analyses. Once you are done with them it would be a good idea to compress/tarball them for archiving. If you are running many assessments it might be a good idea to compress/tarball the results folders that contain many files as each run finishes using the **`-z`**, **`--tarzip`** option.
- Please do take some time to check the log files, these are there for your benefit in order to highlight potential problems that may have occurred during your BUSCO run.
- Compare the results from assessing your data with like-for-like assessments of corresponding publically available data for other closely-related species. In this way, the BUSCO results can be used to claim that your dataset is as good as or better than existing publically available datasets for similar species.
- If manual curation of annotated gene sets was performed, report BUSCO results before and after curation to quantify improvements.

Reporting BUSCO:

- Report results in simple BUSCO notation:
  C:89.0%[S:85.8%,D:3.2%],F:6.9%,M:4.1%,n:3023
- Use the **`BUSCO_plot.py`** script to produce simple graphical summaries (that are easily customisable) for your publication's supporting online information.
- Report the BUSCO, BLAST+, HMMER, and AUGUSTUS versions you used.
- Report the BUSCO set(s) you used for your assessments.
- Report the BUSCO options you used (e.g. starting species for Augustus parameters).
- Report the version(s) of the genome assembly, annotated gene set, or transcriptome that you assessed.

Troubleshooting

Some common issues that might cause errors – please check if correcting any of these might solve any errors you encounter before contacting the BUSCO team.

- Check that your genome, gene set, or transcriptome FASTA format sequences are really in proper FASTA format. Also, some characters might cause problems, like pipes (|) or non-ASCII characters, so to avoid problems try to keep formats simple and clean.
- When running a transcriptome assessment, make sure your input FASTA sequences are indeed DNA sequences and not proteins.
- If you end up with an unexpectedly high number of duplicated BUSCOs from your transcriptome or gene set, please make sure that you **first selected only one transcript/protein per gene** before running your assessments.
- Double check that the dependencies (BLAST+, HMMER, AUGUSTUS) are correctly installed and accessible from the command line for BUSCO to call.
- Check the produced log files to locate the potential source(s) of any error(s).

Please send questions (after reading this user guide) to: support@orthodb.org

Release notes

BUSCO v3.0.2 July 2017

The ~ character can be used in the config.ini file

BUSCO v3.0.1 May 2017

Add the –blast_single_core option to help dealing with the tblastn issue. Add the environment variable BUSCO_CONFIG_FILE to allow custom location for the config.ini file. Minor other fixes.

BUSCO v3.0.0 May 2017

This update focused on refactoring the code to make it more flexible and extendable. It requires BUSCO to be installed through a setup.py mechanism and introduces the use of a config.ini file that makes BUSCO easier to be integrated in a pipeline workflow. It also replaces the GNU GPLv3 license with the MIT license.

BUSCO v2.0.1 March 2017

This minor-update release incorporates additional checks on the status of the tBLASTn step to make sure it has completed correctly. Instead of just producing a warning message, BUSCO will now report an error and the run will terminate. This update has been implemented because it appears that when running on multiple cores, tBLASTn from BLAST+ versions 2.4, 2.5, and 2.6 may sometimes fail to complete. Solutions (for now) are to either roll back to an earlier BLAST+ version, or to run using only a single core.

Acknowledgements

The BUSCO team would like to thank all our enthusiastic users who have contacted us with suggestions to improve the codebase, request new species sets, and beta-test the initial draft versions of BUSCO v2.0 lineages. Many thanks also to everyone who has used BUSCO and cited the publication – we are glad it has proven useful in your research, and a good citation record will help us to obtain future funding to keep developing BUSCO and OrthoDB.

The BUSCO and OrthoDB projects have received funding and infrastructure support from the Foundation Giorgi-Cavaglieria; a Marie Curie International Outgoing Fellowship PIOF-GA-2011-303312; the Schmidheiny Foundation; the Swiss Institute of Bioinformatics SER funding; the Swiss National Science Foundation [31003A_112588, 31003A_125350, 31003A_143936, and PDFMP3_137064]; and the 'Commission Informatique' of the University of Geneva. Some computations were performed at the Vital-IT (http://www.vital-it.ch) centre for high-performance computing of the Swiss Institute of Bioinformatics.