VIROMICS

and Viral Bioinformatics

— A Complete A-Z Practical Roadmap —

From fundamentals and experimental design through a full **Linux analysis pipeline**, **visualization**, and a **public-data mini project** — with **runnable code** and **downloadable scripts**.

- A-Z Practical Roadmap
- Linux-Based Analysis Pipeline
- Visualization & Interpretation
- Real Public Data Mini Project

Dr. Muhammad Aammar Tufail
PhD and PostDoc in Biological Data Science
Senior Bioinformatician
Founder/CEO at codanics.com
Germany

codanics

PRACTICAL & HANDS-ON | RUNTIME CODE & SCRIPTS | DOWNLOADABLE RESOURCES | LEARN. ANALYZE. DISCOVER.

Viromics and Viral Bioinformatics

A Complete A-Z Practical Roadmap

Dr. Muhammad Aammar Tufail

2026-08-17

Table of contents

1 Viromics and Viral Bioinformatics

A Complete A-Z Practical Roadmap

Author — Dr. Muhammad Aammar Tufail. PhD and PostDoc in Biological Data Science · Senior Bioinformatician · Founder/CEO at codanics.com · Germany. Website: codanics.com · YouTube: Codanics.

Download option: After rendering the book, use the PDF button in the sidebar or open `Viromics-and-Viral-Bioinformatics.pdf`.

```
quarto render
```

1.1 Dedication

I dedicate this book to my late father, **Ghulam Mustafa** *(late)* — whose dream it was that his son would help change the way education reaches those who cannot afford it.

When I was in ninth grade, there were days we could not arrange the school fee, and I almost left my studies to support the family. My father would not allow it. *"Beta, never stop learning — I will find a way,"* he said, and through hard labour he kept me in school. That day I promised myself that one day I would build a platform where anyone could learn for free, and only those who could afford it would pay. By the grace of Allah, that platform — Codanics — now teaches hundreds of thousands of learners at no cost. This book is part of that promise, and of his dream.

> ***Please remember my father in your prayers. Allah un ki maghfirat farmaye! Ameen.***

1.2 Who this book is for

This book is for bioinformatics learners who already have the basics and want to specialize in viruses. You will get the most from it if you are comfortable with the Linux terminal, FASTQ and FASTA files, paired-end sequencing, conda or mamba environments, and simple plotting in R or Python. General metagenomics experience helps but is not required.

It suits several readers:

- **students and self-learners** building a portfolio-ready viromics workflow from scratch;
- **wet-lab microbiologists and virologists** who want to analyze their own sequencing data;
- **metagenomics analysts** adding virus-aware steps — viral prediction, CheckV quality, vOTUs, host prediction — to an existing pipeline;
- **instructors** who need a tested, reproducible course with quizzes and downloadable scripts.

If you have never opened a terminal, work through a short Linux primer first, then return here.

1.3 Learning path

The book follows one complete workflow, from first principles to a public-data mini project and research ideas. Read it in order the first time; afterwards, treat each chapter as a standalone reference.

```
flowchart LR
  A[Viromics fundamentals] --> B[Experimental design]
  B --> C[Linux setup]
  C --> V[ViroProfiler: automated pipeline]
  V --> D[Complete manual pipeline]
  D --> E[Visualization and reporting]
  E --> F[Public SRA mini project]
  F --> G[Research project ideas]
```

1.4 What's inside

Nine chapters carry the workflow end to end:

1. **Fundamentals of Viromics** — what a virome is, viral genome types, Baltimore groups, lytic vs. lysogenic lifestyles, and viral dark matter.
2. **Experimental Design** — sampling, VLP enrichment, DNA vs. RNA choices, controls, and how design shapes what you can conclude.
3. **Linux Setup** — building reproducible conda environments and the standard project structure used everywhere in the book.
4. **ViroProfiler: an automated A–Z pipeline** — install, verify databases, and run a complete containerized pipeline on test and real virome data with one command, then interpret every output.
5. **Complete Analysis Pipeline** — the same workflow rebuilt by hand: QC, assembly, viral prediction, CheckV, vOTUs, taxonomy, functional annotation, host prediction, and abundance.
6. **Visualization and Reporting** — turning result tables into figures and an honest, reproducible report.
7. **Public SRA Mini Project** — a full run on real public data, from accession to results.
8. **Practice Quizzes** — self-check questions across the whole workflow.
9. **Project Ideas** — research directions to take the skills further.

Five appendices act as quick references:

- **Appendix A — Linux cheatsheet** for the commands used most often.
- **Appendix B — Tool summary** table with purposes, inputs, outputs, and install commands.
- **Appendix C — Troubleshooting** guide in Problem → Fix format.
- **Appendix D — Glossary** of key terms.
- **Appendix E — Figure generation guide** for the book's illustrations.

1.5 How to use this book

Run the commands in order, and change one thing at a time when something breaks — Appendix C lists the fixes for the most common failures.

You do not have to type everything by hand. Ready-made, reusable scripts live in the `scripts/` folder and can be downloaded from the download buttons inside each chapter — for example `setup_project.sh` to create the standard folders, `check_viromics_installation.sh` to verify your environments, `run_phase4_main_pipeline.sh` for the full pipeline, and `make_figures.R` / `make_figures.py` for the plots. One-command conda environment files are in `envs/`, and a tiny synthetic dataset for practicing the commands is in `data/toy/`.

Getting the files in the PDF edition. The clickable download buttons appear in the HTML edition. In this PDF, every script is in the book's `scripts/` folder, every conda environment file in `envs/`, the toy dataset in `data/toy/`, and example result tables in `data/example/` — download them from the source repository at https://github.com/codanics/viromics-book.

Small example result tables live in `data/example/` (for instance `checkv_quality_summary.tsv`, `votu_abundance.tsv`, `taxonomy.tsv`, and `host_prediction.tsv`). Use them to test the visualization and reporting steps before you commit hours to downloading databases or assembling your own data.

Read the warnings before downloading large databases — some are hundreds of GB.

1.6 Codanics links

- Website: https://www.codanics.com
- Bioinformatics course: https://codanics.com/courses/bioinformatics-ka-chilla/
- YouTube: https://www.youtube.com/%40codanics
- Facebook: https://www.facebook.com/aammar.tufail
- Instagram: https://www.instagram.com/aammartufail/
- TikTok: https://www.tiktok.com/%40draammar

i Hardware assumptions

The runtime and storage estimates in this book assume a workstation with **12 CPU threads, 32 GB RAM, Ubuntu Linux, and SSD storage**. Real runtimes vary with read depth, internet speed, disk speed, and database size.

Preface

This book is developed for Codanics learners and is launched through https://www.codanics.com.

Viromics is the sequencing-based study of viral communities. It sits at the meeting point of biology, NGS, Linux, statistics, ecological interpretation, and careful reporting. Unlike bacterial amplicon profiling, viromics has no universal marker like 16S rRNA. That single fact reshapes the whole workflow: identification, quality control, clustering, and taxonomy all have to be rebuilt around evidence rather than a barcode.

The learner must think like a detective. A viral contig is not accepted because one tool flags it. Confidence grows when several independent lines of evidence agree — viral prediction, hallmark genes, CheckV quality, coverage, taxonomy, functional annotation, and biological context. Where the evidence is thin, the honest answer is an uncertain one, and this book teaches you to say so.

How this edition is organized

This edition moves in one continuous arc: it builds concepts first, then a clean computing environment, then a complete analysis pipeline, and finally visualization, a public-data mini project, and research ideas. Every chapter opens with clear learning objectives and closes with quizzes, and reusable scripts plus small example datasets let you practice each step without waiting on large downloads. Four reference appendices — a Linux cheatsheet, a tool summary, a troubleshooting guide, and a glossary — are meant to be consulted out of order whenever you need them.

What you need before starting

You should be comfortable with:

- Linux terminal basics
- FASTQ files
- paired-end sequencing
- conda or mamba
- FASTA files
- basic metagenomics
- simple R or Python plotting

Hardware assumptions

All runtime and storage estimates in this book use this reference machine:

Resource	Reference value
CPU	12 threads
RAM	32 GB
Storage	SSD
OS	Ubuntu 22.04 or 24.04
Internet	stable broadband

> **Warning**
>
> Tool databases can be much larger than the example dataset. Keep databases on a large disk. For full DRAM, iPHoP, geNomad, and CheckV installations, plan hundreds of GB if you want all databases locally.

Citation style

The book uses Quarto citations from `references.bib`. Core references include CheckV (Nayfach et al. 2021), VirSorter2 (Guo et al. 2021), geNomad (Camargo et al. 2024), vConTACT2 (Bin Jang et al. 2019), VIBRANT (Kieft, Zhou, and Anantharaman 2020), DRAM-v (Shaffer et al. 2020), eggNOG-mapper (Cantalapiedra et al. 2021), MEGAHIT (Li et al. 2015), fastp (Chen et al. 2018), MAFFT (Katoh and Standley 2013), IQ-TREE 2 (Minh et al. 2020), and ICTV taxonomy resources (International Committee on Taxonomy of Viruses 2026).

2 Fundamentals of Viromics

Viromics uses sequencing and bioinformatics to study the **virome** — the viral community associated with a sample or environment. If a microbial sample is a city, then:

- **virology** studies one resident in detail (how a single virus infects and replicates);
- **metagenomics** surveys every resident's genetic material at once; and
- **viromics** focuses on the viral population — who the viruses are, what they can do, and which hosts they interact with.

That shift in focus changes almost everything downstream, from how you prepare a library to how you decide a contig is really viral.

Learning objectives — by the end of this chapter you will be able to:

- distinguish viromics from classical virology and general metagenomics;
- compare the major viral genome types and Baltimore groups;
- explain why viral genomes require different bioinformatics strategies from bacterial genomes;
- read the sequence signals that separate lytic infection from temperate lifestyles; and
- use Linux commands to filter and summarize a tab-separated dataset.

> **!** A virome is an operational measurement
>
> No sequencing protocol captures every virus equally well. The observed virome depends on sampling, particle enrichment, nucleic-acid extraction, DNA or RNA selection, library preparation, sequencing depth, and the analysis workflow. A reported virome is therefore the viral community **detected under a particular protocol**, not a complete inventory of every virus present.

2.1 Viromics, virology, and metagenomics

Field	Main question	Typical data	City analogy
Virology	How does a virus infect, replicate, and affect its host?	isolates, cultures, PCR assays, microscopy	one resident, studied closely
Metagenomics	What genetic material is present in a community?	total-community DNA or RNA	every resident at once

Field	Main question	Typical data	City analogy
Viromics	Which viruses and viral functions are represented in a community?	virus-enriched data or viral sequences from metagenomes	the viral population

Unlike bacteria and archaea, viruses do not share a universal marker gene comparable to 16S rRNA. Viral sequence identification therefore combines several imperfect signals: viral hallmark genes, similarity to reference sequences, gene density and orientation, sequence composition, genome architecture, read coverage, and host-association evidence (Minot et al. 2011). No single signal is sufficient in every case, especially for short contigs and highly novel viruses.

2.2 Viral genome types

Viral genomes may consist of DNA or RNA and may be single- or double-stranded. They can also be linear or circular, and segmented or non-segmented. These properties affect library preparation, assembly, annotation, and the databases used for classification.

Genome type	Meaning	Example
dsDNA	double-stranded DNA	many bacteriophages
ssDNA	single-stranded DNA	microviruses
dsRNA	double-stranded RNA	reoviruses
+ssRNA	positive-sense RNA	coronaviruses
-ssRNA	negative-sense RNA	influenza viruses
+ssRNA-RT	positive-sense RNA with reverse transcription	retroviruses
dsDNA-RT	double-stranded DNA with reverse transcription	hepadnaviruses

2.3 Virus genomes versus bacterial genomes

Many viral genomes are compact and encode functions that help the virus enter a host, replicate, assemble particles, evade host defenses, and spread. Bacterial genomes are generally larger and encode the systems needed to maintain a cellular organism, including translation, energy generation, membrane transport, and stress responses. These are useful trends rather than universal rules: giant viruses can have large genomes, and some viruses carry auxiliary metabolic genes that redirect host metabolism during infection.

Feature	Viral genome	Bacterial genome	Example
Typical genome size	a few kb to hundreds of kb, with exceptions above 1 Mb	hundreds of kb to several Mb	SARS-CoV-2 is about 30 kb; *E. coli* K-12 is about 4.6 Mb

Feature	Viral genome	Bacterial genome	Example
Nucleic acid type	DNA or RNA	DNA	influenza has RNA, T4 phage has dsDNA
Strandedness	single or double stranded	typically double stranded	ssDNA phages exist, bacteria are generally dsDNA
Shape	linear, circular, segmented, non-segmented	usually circular chromosome, sometimes multiple replicons	hepatitis B partially dsDNA, *Vibrio* has two chromosomes
Gene density	very high	moderate	viral genomes often have overlapping genes
Ribosomes	absent	present in the cell	viruses use host ribosomes
Metabolic capacity	none independent; some encode metabolic genes	extensive cellular metabolism	viruses redirect host resources
Reproduction	depends on a host cell	growth followed by cell division	phages replicate only in suitable hosts

2.3.1 Genome structure examples

- **T4 bacteriophage**: a tailed bacteriophage with a linear dsDNA genome.
- **PhiX174**: a bacteriophage with a small, circular ssDNA genome; also widely used as an Illumina sequencing control.
- **SARS-CoV-2**: positive-sense ssRNA, about 30 kb, one of the largest RNA virus genomes.
- **Influenza A virus**: negative-sense ssRNA with a segmented genome.
- ***Escherichia coli***: a bacterium with a circular chromosome and thousands of genes that support cellular life.

These differences explain why viral genome interpretation often emphasizes hallmark genes, genome architecture, terminal repeats, coverage patterns, and host association. Bacterial genome interpretation can additionally draw on broadly conserved genes, metabolic pathways, and taxonomic markers. The Baltimore classification complements sequence-based taxonomy by organizing viruses according to how they produce mRNA (Baltimore 1971).

2.4 Baltimore classification

The Baltimore system asks a practical question: **how does the virus produce positive-sense mRNA that host ribosomes can translate?** The answer highlights which polymerases or reverse-transcription steps are required during infection.

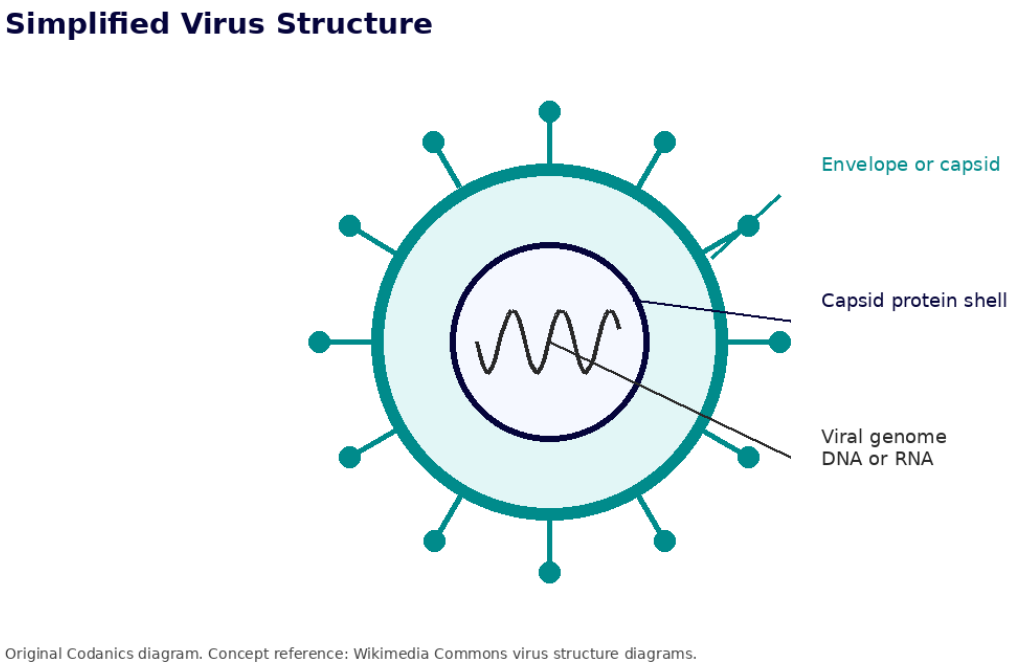


Figure 2.1: Simplified virus structure showing the viral genome, capsid, and optional envelope.

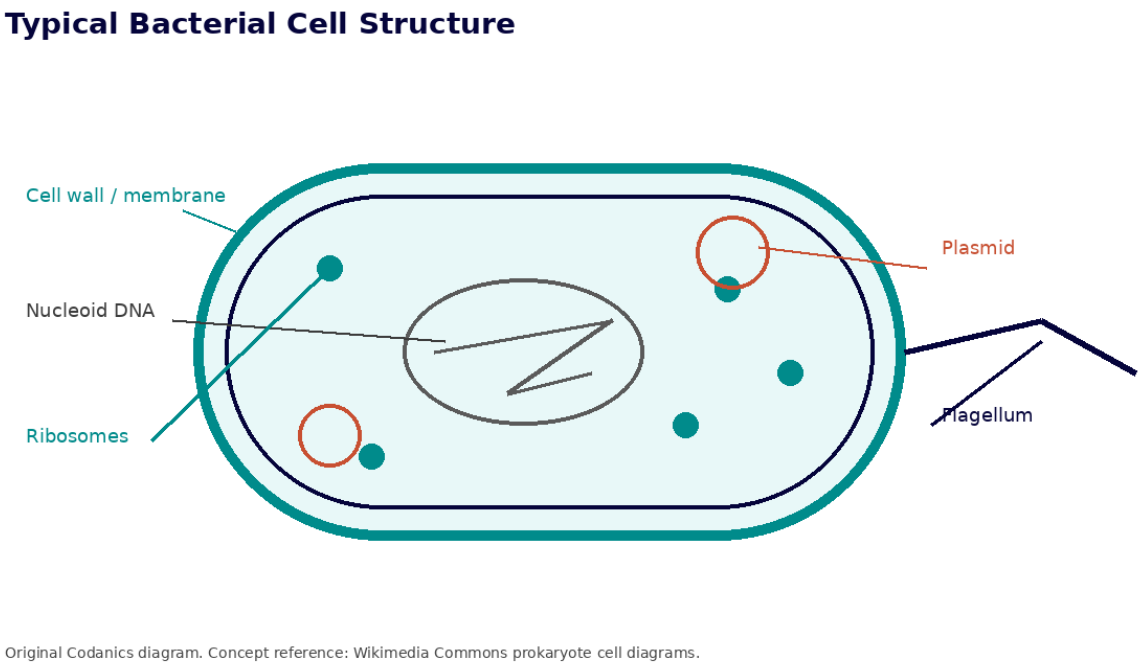


Figure 2.2: A typical prokaryotic cell. Comparing an independent bacterial cell with a host-dependent virus highlights why viruses lack ribosomes and independent metabolism.

A cooking analogy for Baltimore groups

Think of mRNA as a recipe the host ribosome can cook from directly:

- **+ssRNA (IV)** arrives as a ready-to-use recipe — translate immediately.
- **-ssRNA (V)** arrives written backwards; a viral polymerase must first rewrite it forwards.
- **dsRNA (III)** comes as a sealed double recipe; a viral polymerase transcribes a usable copy.
- **dsDNA (I)** brings a whole cookbook that host machinery transcribes.
- **ssDNA (II)** is half a cookbook that is first completed into dsDNA.
- **Retroviruses (VI)** carry an RNA recipe they reverse-transcribe into a DNA cookbook first.
- **dsDNA-RT (VII)** keep a DNA cookbook but copy it through an RNA intermediate.

Group	Genome	Route to mRNA
I	dsDNA	dsDNA is transcribed into mRNA
II	ssDNA	ssDNA is converted to dsDNA, then transcribed
III	dsRNA	viral RNA-dependent RNA polymerase produces mRNA
IV	+ssRNA	the genome can function directly as mRNA
V	-ssRNA	viral RNA-dependent RNA polymerase produces +mRNA
VI	+ssRNA-RT	RNA is reverse-transcribed into DNA, then transcribed
VII	dsDNA-RT	repaired dsDNA is transcribed; an RNA intermediate is reverse-transcribed during replication

Classification systems answer different questions

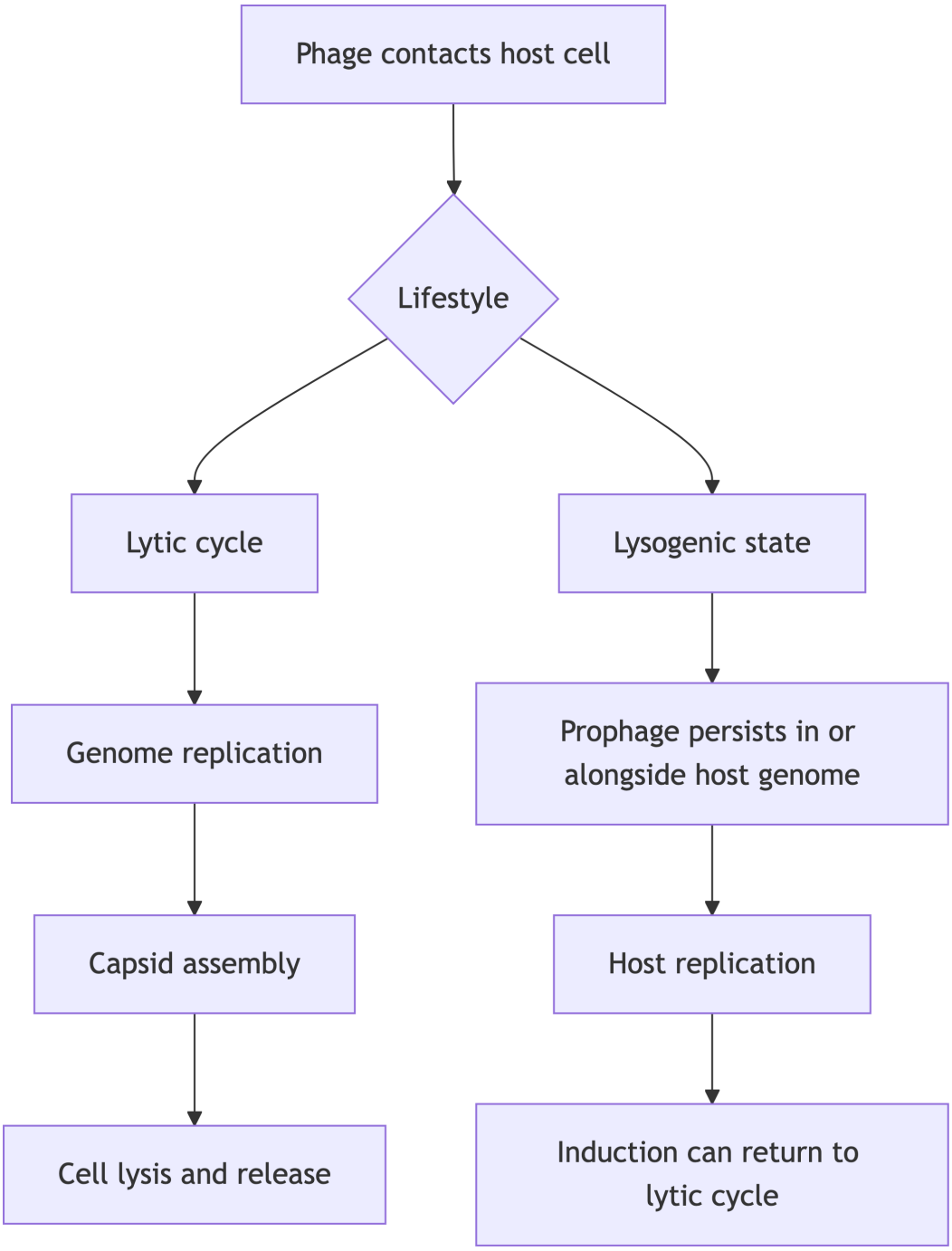
Baltimore groups describe the route to mRNA; they are not taxonomic ranks. Formal virus taxonomy instead organizes viruses into realms, kingdoms, phyla, classes, orders, families, genera, and species. Two viruses can share a Baltimore group without being close evolutionary relatives.

2.5 Lytic and lysogenic lifestyles

During a **lytic** infection, a phage replicates, assembles new particles, and releases progeny — often by lysing the host cell. A **temperate** phage can instead enter **lysogeny**, in which its genome persists as a **prophage** integrated into the host chromosome or maintained as an extrachromosomal

element. Stress or other signals may induce the prophage to resume productive infection (Roux et al. 2015).

```
flowchart TD
  A[Phage contacts host cell] --> B{Lifestyle}
  B --> C[Lytic cycle]
  C --> D[Genome replication]
  D --> E[Capsid assembly]
  E --> F[Cell lysis and release]
  B --> G[Lysogenic state]
  G --> H[Prophage persists in or alongside host genome]
  H --> I[Host replication]
  I --> J[Induction can return to lytic cycle]
```



Why does this matter for bioinformatics? Because each lifestyle leaves different fingerprints in your data:

Lifestyle	Expected sequence signals
Lytic / free particles	free viral genomes assembling as separate contigs, circular or complete genomes, high read coverage, hallmark genes such as capsid, terminase, portal, and polymerase

Lifestyle	Expected sequence signals
Lysogenic / prophage	viral regions flanked by host DNA, integrase or recombinase genes, attachment (*att*) sites, mixed host–virus signals on one contig

Recognizing these signals explains why prophages can appear *inside* bacterial contigs and why tools such as CheckV trim host flanks from proviruses.

2.6 Viromes across environments

Environment	Typical viral signals	Bioinformatics challenge
Human gut	bacteriophages, eukaryotic viruses	high inter-individual variation
Soil	phages, plant viruses, fungal viruses	inhibitors and very high diversity
Marine	marine phages	huge diversity and novelty
Wastewater	mixed human, animal, bacterial viruses	mixed sources and contamination
Plant	plant RNA viruses, phages, mycoviruses	host RNA or DNA background
Animal	wildlife and livestock viruses	zoonotic surveillance context

Phages are not passive passengers. They drive microbial mortality, nutrient cycling, and horizontal gene transfer, and in the human body they form distinct sub-viromes of the gut, oral cavity, skin, and respiratory tract.

2.7 Why viral dark matter matters

Many viral sequences have no close match in current reference databases — a knowledge gap often called **viral dark matter**. Reference-only searches therefore miss or weakly classify many viral contigs. Modern workflows combine complementary approaches: tools such as VirSorter2 (Guo et al. 2021) and geNomad (Camargo et al. 2024) identify candidate viral sequences, while CheckV estimates completeness and contamination of assembled viral genomes (Nayfach et al. 2021). Predictions remain hypotheses whose confidence depends on contig length, database coverage, and agreement among multiple signals.

When a reference match is missing, these clues still help build confidence that a contig is viral:

- viral hallmark genes (capsid, large terminase subunit, portal, and polymerase);
- high gene density with few cellular housekeeping genes;
- circularity or direct terminal repeats;
- CRISPR spacer matches to a known host;
- consistent, even read coverage; and
- absence of ribosomal RNA and other cellular markers.

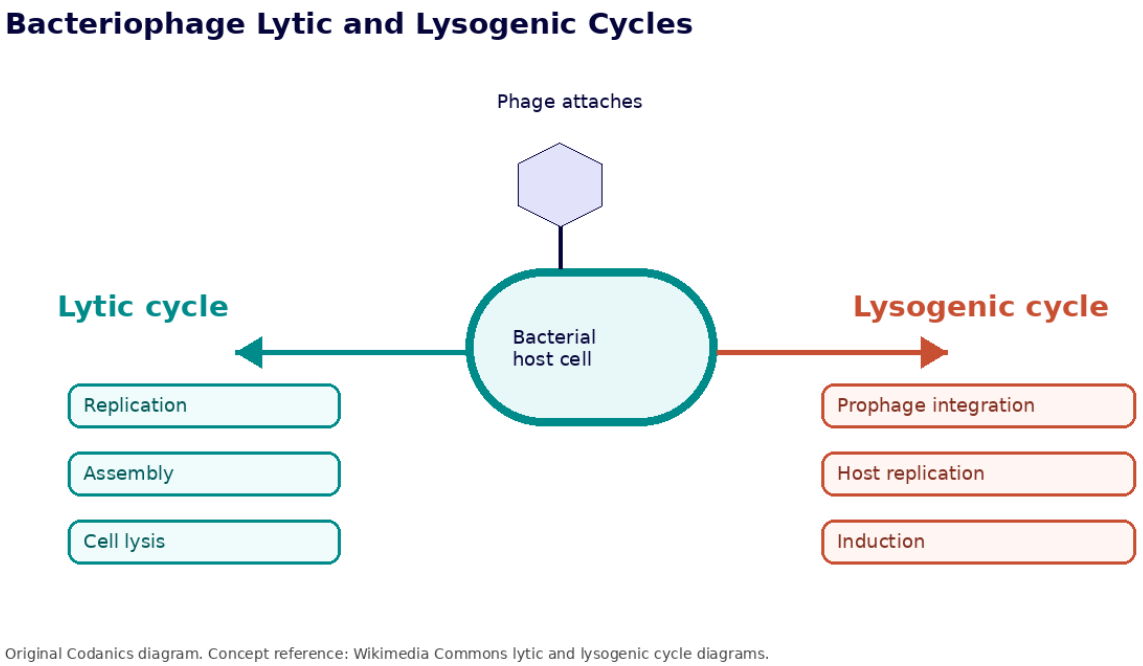


Figure 2.3: Lytic and lysogenic bacteriophage cycles.

2.7.1 Why viromics is harder than bacterial metagenomics

Challenge	Why it complicates analysis
No universal marker gene	there is no viral 16S; identification needs multiple signals
Enormous diversity	viral sequence space is vast and largely uncharted
Small genomes	short contigs carry less evidence per sequence
Reference bias	databases are incomplete, so novel viruses are missed
Contamination	host and bacterial DNA can dominate a library
Host DNA background	plant, animal, or human reads can swamp viral reads
RNA vs DNA workflows	RNA viromes need reverse transcription and are less stable
Prophages	integrated viruses blur the host–virus boundary

2.8 Linux activity: explore genome types

This activity creates a small tab-separated values (TSV) file, selects RNA genome types, and summarizes the genome categories. It requires only a POSIX-like shell and standard command-line

tools. First, set up the standard project folders you will reuse throughout the book.

```
# One-time: create the standard project structure used across the book
bash setup_project.sh          # or: bash ~/Downloads/setup_project.sh
```

2.8.1 1. Create the dataset

```
mkdir -p ~/viromics_course/phase1_fundamentals
cd ~/viromics_course/phase1_fundamentals

cat > viral_genome_types.tsv << 'EOF'
Virus   Group   Genome  Example
T4_phage    I   dsDNA   Bacteriophage
Microvirus  II  ssDNA   Bacteriophage
Rotavirus   III dsRNA   Animal_virus
Coronavirus IV  +ssRNA  Animal_virus
Influenza   V   -ssRNA  Animal_virus
HIV VI  +ssRNA-RT   Retrovirus
Hepatitis_B VII dsDNA-RT    Hepadnavirus
EOF

column -t -s $'\t' viral_genome_types.tsv   # optional pretty printing
```

2.8.2 2. Select RNA genome types

```
awk -F'\t' 'NR==1 || $3 ~ /RNA/' viral_genome_types.tsv
```

`-F'\t'` tells `awk` that fields are separated by tabs. `NR==1` retains the header, and `$3 ~ /RNA/` selects records whose third field contains `RNA`.

2.8.3 3. Count genome categories

```
cut -f3 viral_genome_types.tsv | tail -n +2 | sort | uniq -c
```

The pipeline extracts the third column, removes the header, sorts identical values together, and counts them. Because this teaching dataset contains one example per Baltimore group, each category should have a count of one.

Check your understanding

Modify the `awk` command to select bacteriophages using the fourth column:

```
awk -F'\t' 'NR==1 || $4 == "Bacteriophage"' viral_genome_types.tsv
```

Exercise: build an environments table

Create `virome_environments.tsv` with columns `Environment`, `Main_viruses`, `Main_hosts`, and `Bioinformatics_challenge` for five environments (human gut, soil, marine, wastewater, plant), then print only the rows where the challenge mentions "diversity".
One solution:

```
cat > virome_environments.tsv << 'EOF'
Environment Main_viruses    Main_hosts  Bioinformatics_challenge
Human_gut   Bacteriophages  Gut_bacteria    high_individual_variation
Soil    Phages_and_plant_viruses    Soil_microbes   inhibitors_and_high_diversity
Marine  Marine_phages   Marine_bacteria huge_diversity_and_novelty
Wastewater  Mixed_viruses   Mixed_hosts mixed_sources_contamination
Plant   Plant_RNA_viruses   Plant_cells host_background_signal
EOF

awk -F'\t' 'NR==1 || $4 ~ /diversity/' virome_environments.tsv
```

Estimated resources: one CPU thread, less than 100 MB RAM, less than 1 MB storage, and under one minute.

Common beginner mistakes

- **Trusting one tool.** A single positive call is a hypothesis, not a conclusion — require agreement across evidence.
- **Ignoring CheckV quality.** Reporting fragments as if they were complete genomes overstates your results.
- **Forcing species-level names.** For novel viruses, family-level or higher assignments are usually more honest.

2.9 Key takeaways

- Viromics targets the viral *population* of a sample, sitting between single-virus virology and whole-community metagenomics, and every reported virome is an operational, protocol-dependent measurement rather than a complete inventory.
- Viral genomes span DNA and RNA, single- and double-stranded, linear, circular, and segmented forms; the Baltimore system organizes them by their route to translatable mRNA and complements formal taxonomy (Baltimore 1971).
- Viruses lack a universal marker gene like 16S rRNA, so identification must combine several

imperfect signals — hallmark genes, gene density, composition, coverage, and host association — rather than relying on any single line of evidence (Minot et al. 2011).
- Lytic and lysogenic lifestyles leave distinct fingerprints in sequence data, which is why prophages appear embedded in host contigs and why proviral host flanks must be trimmed before analysis.
- Much of the virosphere is uncharacterized "viral dark matter," so predictions from tools such as VirSorter2 remain hypotheses whose confidence rises with contig length, database coverage, and agreement across methods (Guo et al. 2021; Nayfach et al. 2021).

2.10 Further reading

- Baltimore (1971) introduces the classification of viruses by their strategy for producing mRNA, the conceptual backbone of the Baltimore groups discussed here.
- Minot et al. (2011) is a foundational human gut virome study illustrating why viral identification depends on multiple, imperfect signals.
- Nayfach et al. (2021) describes CheckV, the standard for estimating completeness and contamination of assembled viral genomes and for trimming prophage host flanks.
- The ICTV maintains the authoritative framework for virus taxonomy, complementing Baltimore groups (International Committee on Taxonomy of Viruses 2026); see the official site at https://ictv.global.

2.11 Chapter figure

Generate this figure

Save as: `images/ch01-viromics-ecosystem.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean educational infographic showing viromics as a viral population inside a microbial ecosystem. On the left, show a stylized ecosystem containing bacteria, archaea, fungi, plant cells, and animal cells, with bacteriophages, RNA viruses, and DNA viruses highlighted among them. In the center, show viral particles being enriched and their genomes extracted. On the right, show viral contigs flowing into a Linux terminal representing a bioinformatics pipeline. Use a modern scientific style with teal and navy Codanics branding and clear labels.

2.12 Quiz: Fundamentals

Q1. What is a virome?

A. all bacteria in a sample B. all viruses in a sample or ecosystem C. only disease-causing viruses D. only RNA viruses

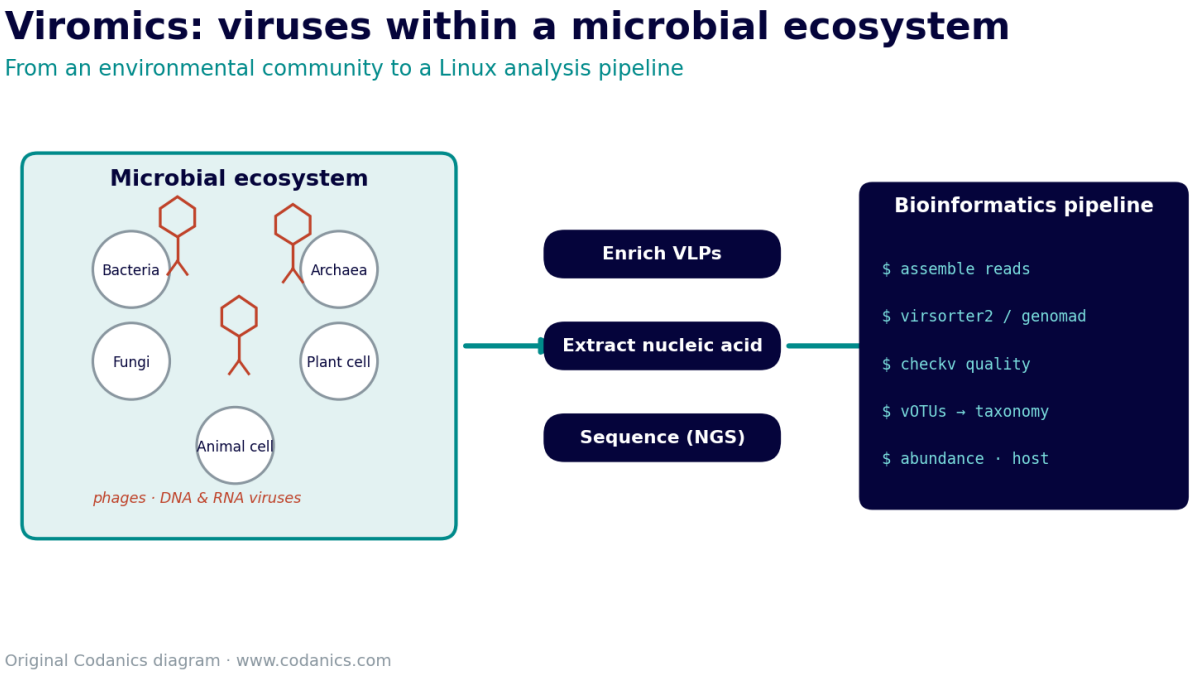


Figure 2.4: Viromics places viruses inside a microbial ecosystem and feeds the resulting sequences into a Linux-based bioinformatics pipeline.

> **Note**
>
> **Answer:** B. A virome is the viral community associated with a sample or environment, although an experiment generally detects only a protocol-dependent subset.

Q2. Why is viromics difficult compared with 16S profiling?

A. viruses have no universal marker gene B. viruses are always larger than bacteria C. viral genomes are all identical D. viruses cannot be assembled

> **Note**
>
> **Answer:** A. There is no universal viral barcode equivalent to 16S rRNA.

Q3. What is a prophage?

A. a temperate phage genome persisting in or alongside a host genome B. a sequencing adapter C. a protein domain D. a host ribosome

> **Note**
>
> **Answer:** A. A prophage persists in a lysogen, commonly integrated into the host chromosome but sometimes maintained extrachromosomally.

Q4. Which Baltimore group includes +ssRNA viruses?

A. I B. II C. IV D. VII

> **Note**
>
> **Answer:** C. Group IV includes positive-sense single-stranded RNA viruses.

Q5. What is viral dark matter?

A. viral sequences with no close reference match B. contaminated adapters C. low-quality bases only D. a type of culture medium

> **Note**
>
> **Answer:** A. Many viral sequences remain poorly represented in reference databases.

2.13 Interactive quiz: Fundamentals

3 Experimental Design for Viromics

A viromics project begins before the first command. In viromics, **experimental design is more important than the pipeline** — a good pipeline cannot rescue a poorly planned virome experiment. The research question controls the sample type, enrichment strategy, sequencing platform, controls, and analysis plan.

Viromics is like fishing in a huge ocean

- Your **sample type** decides *where you fish*.
- Your **enrichment method** decides *which net you use*.
- Your **sequencing platform** decides *how clearly you can see the fish*.
- Your **controls** tell you whether you caught real fish or plastic that fell off your own boat.

Every design choice below is really one of these four decisions.

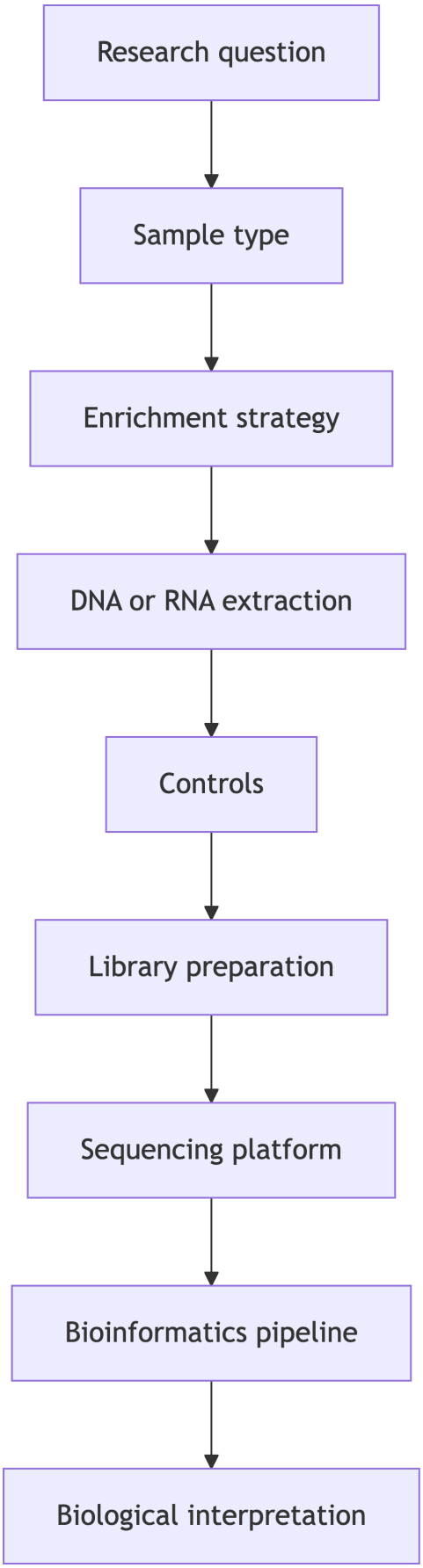
Learning objectives — by the end of this chapter you will be able to:

- turn a vague aim into a strong, testable viromics research question;
- choose a sample type, enrichment strategy, and DNA or RNA workflow for a given goal;
- explain how each enrichment step biases the observed community and how to report it;
- select a sequencing platform and the right set of negative and positive controls;
- plan biological versus technical replication for a defensible study; and
- build and validate a sample metadata sheet and FASTQ manifest on the command line.

3.1 Design workflow

Never start with the software. Start with the biological question and let it drive every downstream choice.

```
flowchart TD
  A[Research question] --> B[Sample type]
  B --> C[Enrichment strategy]
  C --> D[DNA or RNA extraction]
  D --> E[Controls]
  E --> F[Library preparation]
  F --> G[Sequencing platform]
  G --> H[Bioinformatics pipeline]
  H --> I[Biological interpretation]
```


Research question
Sample type
Enrichment strategy
DNA or RNA extraction
Controls
Library preparation
Sequencing platform
Bioinformatics pipeline
Biological interpretation

3.2 Start with a strong question

A weak question is too broad to design around:

```
What viruses are in my sample?
```

A strong question defines the environment, host, comparison, nucleic-acid type, and expected interpretation:

```
How does nitrogen fertilization change the DNA virome of maize
rhizosphere soil across crop growth stages?
```

The stronger version already tells you the sample type (soil), the host context (maize rhizosphere), the treatment (nitrogen fertilization), the comparison (across growth stages), and the analysis direction (differential DNA virome).

Field	Good viromics question
Human gut	How does antibiotic treatment affect gut bacteriophage diversity?
Soil	How do cropping systems change soil viral communities?
Plant pathology	Which RNA viruses are associated with symptomatic tomato leaves?
Wastewater	Can wastewater viromics detect enteric RNA viruses across seasons?
Animal surveillance	Which viruses are present in bat fecal samples from different habitats?

3.3 Sample types

Different sample types carry different viruses and different problems.

Sample type	Common viruses detected	Main challenge
Feces / gut	bacteriophages, enteric viruses	host and bacterial background, inhibitors
Soil	phages, plant viruses, fungal viruses	complex matrix, humic acids, low viral recovery
Water / wastewater	RNA and DNA viruses, phages	dilution, concentration required
Plant leaves	plant RNA viruses, phages, mycoviruses	plant RNA and DNA background
Blood / plasma	eukaryotic viruses, phages	very low biomass, contamination sensitive
Respiratory swabs	RNA viruses, bacteriophages	host RNA dominance

Sample type	Common viruses detected	Main challenge
Animal tissues	host-associated viruses	host genome background

Low-biomass samples such as plasma, respiratory swabs, and some viral-like particle preparations are especially vulnerable to contamination, because even small amounts of reagent or environmental nucleic acid can become a large fraction of the final read set after amplification and sequencing. These samples need strict negative controls.

3.4 Enrichment methods

A virome sample contains far more non-viral than viral material — host cells, bacteria, fungi, archaea, free nucleic acids, and food, soil, or plant particles. Enrichment increases the proportion of viral particles before sequencing.

Method	Purpose	What it removes or enriches
Centrifugation	remove large debris and cells	plant tissue, soil particles, host cells
Filtration	remove cells larger than viruses	bacteria, eukaryotic cells
Nuclease treatment	degrade free DNA/RNA outside capsids	host and bacterial free nucleic acids
Viral concentration	increase viral particles	useful for water and wastewater
Density gradient	purify viral particles	selective but laborious
Random amplification	boost low-input viral nucleic acid	useful but may introduce bias

3.4.1 Enrichment is not neutral: what each step loses

Enrichment improves viral signal but changes the original community structure. Teach the trade-offs explicitly.

A **0.22 µm filter** removes most bacteria, but may also lose:

- giant viruses,
- viruses attached to cells,
- viruses attached to particles, and
- fragile enveloped viruses.

Nuclease treatment reduces free host DNA and RNA, but may lose:

- damaged viral particles,
- unprotected viral genomes, and
- fragile RNA viruses.

Random amplification raises input from low-biomass samples, but may create:

- uneven genome coverage,
- overrepresentation of short templates,
- chimeric artifacts, and
- artificial abundance bias.

> **Warning**
>
> Enrichment improves viral signal, but it does not preserve the original community perfectly. Always report the enrichment method clearly so results can be interpreted and compared.

3.5 DNA virome or RNA virome

Choosing DNA or RNA is one of the most important design decisions, because it changes extraction, library preparation, and interpretation.

- A **DNA virome** targets dsDNA and ssDNA phages, archaeal and eukaryotic DNA viruses, and prophage-like sequences when total DNA is used.
- An **RNA virome** targets +ssRNA, -ssRNA, and dsRNA viruses, including most plant, animal, and clinically important viruses.

Feature	DNA virome	RNA virome
Starting molecule	DNA	RNA
Stability	more stable	less stable
Extraction	DNA extraction	RNA extraction
Reverse transcription	not required	required
Common targets	phages, DNA viruses	plant / animal / human RNA viruses
Main risk	host or bacterial DNA background	RNA degradation
Library prep	DNA library	cDNA library
Bioinformatics	DNA viral contigs	RNA viral contigs / transcripts

Use **DNA viromics** for phage ecology (soil, gut, marine), DNA viruses, viral dark matter, and prophages from microbial assemblies. Use **RNA viromics** for plant viral disease, respiratory and enteric viruses, zoonotic and wastewater RNA surveillance, and emerging-virus discovery. If you need both, design **separate DNA and RNA workflows** rather than forcing one library to do both jobs.

3.6 Total metagenome versus virus-enriched virome

Total metagenomes keep microbial context and prophages but can have very low viral read fractions, so viral assembly may be poor. Virus-enriched viromes improve viral recovery and low-abundance detection but lose cell-associated viruses, distort abundance, and can amplify contaminants. The

right choice follows the question: host–virus association favours total metagenomes; viral community structure favours enrichment.

3.7 Sequencing platform

Illumina gives short, highly accurate reads and is the workhorse for community comparison and abundance. Nanopore gives long reads that can span whole viral genomes and enables rapid field work, at the cost of higher per-read error and less straightforward abundance estimation. Hybrid designs combine both: Nanopore for contiguity, Illumina for accuracy.

Goal	Recommended platform
Many samples, high accuracy	Illumina
Complete viral genomes	Nanopore or hybrid
RNA virus surveillance	Illumina or Nanopore
Low-input clinical virome	Illumina with strong controls
Rapid field detection	Nanopore
Publication-grade community comparison	Illumina, often with validation
Genome closure	Hybrid Illumina plus Nanopore

3.8 Controls

Controls are not optional; they are part of the bioinformatics interpretation. Clean negative controls should produce very few reads — many reads in a blank signal a major contamination risk.

Control	Purpose
Extraction blank	detect contamination from extraction kit and reagents
Library blank	detect contamination from library preparation
No-template control	detect PCR and amplification contamination
Mock viral community	check pipeline recovery and bias
Spike-in control	measure extraction and sequencing performance
Positive control	confirm the protocol can detect known viruses
Technical replicate	check sequencing and library reproducibility
Biological replicate	capture real biological variation

3.9 Common contamination sources

Viral metagenomics is especially sensitive to contamination, because unknown viral-like sequences may be misread as real biology when controls are missing.

Source	Example
Reagents	extraction kit nucleic acids
Lab environment	aerosols, previous PCR products
Cross-sample contamination	splashing, pipetting
Index hopping	reads assigned to the wrong barcode or index
Barcode cross-talk	multiplexing artifact
Host DNA / RNA	human, plant, or animal background
Bacterial DNA	incomplete filtration or lysis
PhiX	Illumina control spike-in
Adapter contamination	library-prep artifacts

3.10 Replication strategy

Biological replicates are different biological samples — soil from plot 1, plot 2, plot 3. They answer: *is this pattern real in nature?* **Technical replicates** are repeated processing or sequencing of the same sample — one DNA extract split into two library preps. They answer: *is the method reproducible?*

For student and small projects:

- **Minimum:** 3 biological replicates per group.
- **Better:** 5 or more biological replicates per group.
- **Always:** include negative controls.

3.10.1 Minimum teaching design

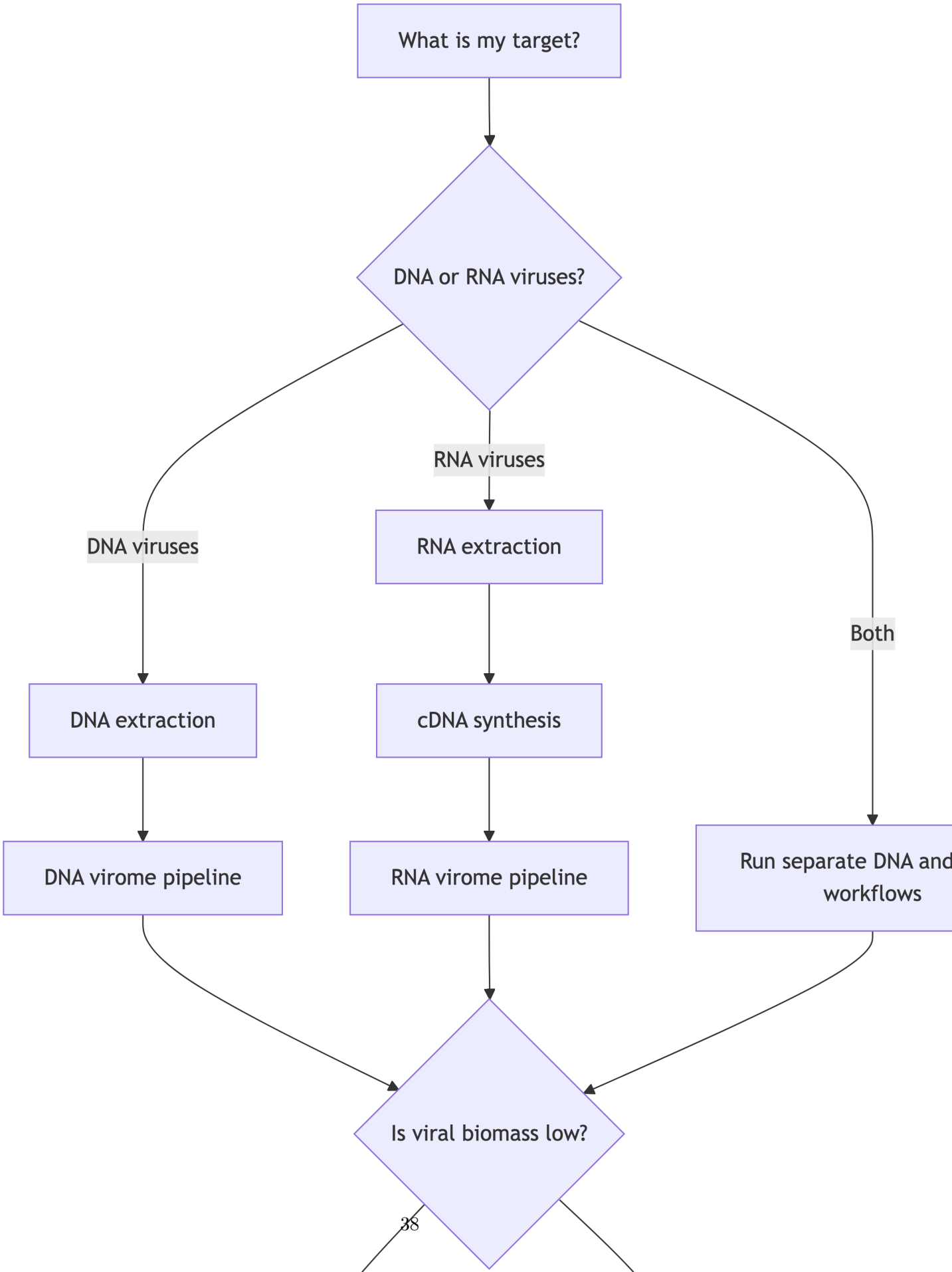
A defensible small project comparing two soil treatments:

Component	Count
Treatment A biological replicates	3
Treatment B biological replicates	3
Extraction blank	1
Library blank	1
Total libraries	**8**

That is `3 + 3 + 2 controls = 8 libraries`, sequenced as Illumina paired-end 150 bp, with a mock viral community added if available.

3.11 A decision tree for planning

```
flowchart TD
  A[What is my target?] --> B{DNA or RNA viruses?}
  B -->|DNA viruses| C[DNA extraction] --> D[DNA virome pipeline]
  B -->|RNA viruses| E[RNA extraction] --> F[cDNA synthesis] --> G[RNA virome pipeline]
  B -->|Both| H[Run separate DNA and RNA workflows]
  D --> I{Is viral biomass low?}
  G --> I
  H --> I
  I -->|Yes| J[Add extraction and library blanks, spike-ins, contamination analysis]
  I -->|No| K[Standard controls]
  J --> L{Do I need complete genomes?}
  K --> L
  L -->|Yes| M[Nanopore or hybrid sequencing]
  L -->|No| N{Many samples and abundance comparison?}
  N -->|Yes| O[Illumina is usually practical]
  N -->|No| O
```


What is my target?
DNA or RNA viruses?
DNA viruses
RNA viruses
Both
DNA extraction
RNA extraction
cDNA synthesis
DNA virome pipeline
RNA virome pipeline
Run separate DNA and
workflows
Is viral biomass low?

3.12 Metadata is part of the design

A viromics project without metadata is weak. Record at least the sample identity, environment, host, treatment, nucleic-acid type, enrichment, control type, platform, read type, and extraction and library batches. First set up the standard project folders you will reuse across the book.

```
# One-time: create the standard project structure used across the book
bash setup_project.sh          # or: bash ~/Downloads/setup_project.sh
```

3.12.1 Create the Phase 2 workspace

```
mkdir -p ~/viromics_course/phase2_experimental_design
cd ~/viromics_course/phase2_experimental_design
mkdir -p raw_reads metadata controls logs reports design_tables
```

3.12.2 Build the sample metadata sheet

```
cat > metadata/sample_metadata.tsv << 'EOF'
sample_id   environment host    treatment   replicate   nucleic_acid    enrichment  control_typ
Soil_A_01   soil    maize   Treatment_A 1   DNA filtration_nuclease biological_sample   batch1
Soil_A_02   soil    maize   Treatment_A 2   DNA filtration_nuclease biological_sample   batch1
Soil_A_03   soil    maize   Treatment_A 3   DNA filtration_nuclease biological_sample   batch1
Soil_B_01   soil    maize   Treatment_B 1   DNA filtration_nuclease biological_sample   batch1
Soil_B_02   soil    maize   Treatment_B 2   DNA filtration_nuclease biological_sample   batch1
Soil_B_03   soil    maize   Treatment_B 3   DNA filtration_nuclease biological_sample   batch1
EXT_BLANK_01    blank   none    none    NA  DNA none    extraction_blank    batch1  lib1    Ill
LIB_BLANK_01    blank   none    none    NA  DNA none    library_blank   batch1  lib1    Illumin
EOF

column -t -s $'\t' metadata/sample_metadata.tsv
```

3.12.3 Create a FASTQ manifest

A manifest links each sample ID to its read files and prevents sample mix-ups.

```
cat > metadata/fastq_manifest.tsv << 'EOF'
sample_id   forward_read    reverse_read
Soil_A_01   raw_reads/Soil_A_01_R1.fastq.gz raw_reads/Soil_A_01_R2.fastq.gz
Soil_A_02   raw_reads/Soil_A_02_R1.fastq.gz raw_reads/Soil_A_02_R2.fastq.gz
Soil_A_03   raw_reads/Soil_A_03_R1.fastq.gz raw_reads/Soil_A_03_R2.fastq.gz
Soil_B_01   raw_reads/Soil_B_01_R1.fastq.gz raw_reads/Soil_B_01_R2.fastq.gz
Soil_B_02   raw_reads/Soil_B_02_R1.fastq.gz raw_reads/Soil_B_02_R2.fastq.gz
```

```
Soil_B_03   raw_reads/Soil_B_03_R1.fastq.gz raw_reads/Soil_B_03_R2.fastq.gz
EXT_BLANK_01    raw_reads/EXT_BLANK_01_R1.fastq.gz  raw_reads/EXT_BLANK_01_R2.fastq.gz
LIB_BLANK_01    raw_reads/LIB_BLANK_01_R1.fastq.gz  raw_reads/LIB_BLANK_01_R2.fastq.gz
EOF
```

3.12.4 Validate that sample IDs match

```
cut -f1 metadata/sample_metadata.tsv | tail -n +2 | sort > metadata/ids_metadata.txt
cut -f1 metadata/fastq_manifest.tsv | tail -n +2 | sort > metadata/ids_manifest.txt
comm -3 metadata/ids_metadata.txt metadata/ids_manifest.txt
```

`comm -3` prints lines unique to either file. **No output means the IDs match** — a habit worth keeping before every run.

Estimated resources on 12 threads and 32 GB RAM: metadata validation finishes in seconds and uses under 1 MB of storage. When planning raw-data storage, start at 3x the expected compressed FASTQ size, because SRA download, decompression, and intermediate files can temporarily coexist.

⚠ Common mistakes

- **Forgetting negative controls.** In viromics, controls are part of the bioinformatics interpretation, not just wet-lab hygiene. No negative controls means weak contamination interpretation.
- **Mixing DNA and RNA viromes without planning.** They need different extraction, library prep, and interpretation — design them as separate workflows.
- **Treating read abundance as true viral abundance.** Read counts are shaped by genome length, extraction efficiency, amplification bias, library prep, sequencing depth, and enrichment. Use normalized measures such as coverage, RPKM, or TPM, and interpret carefully.

ℹ Exercise: design a plant RNA virome study

Create a metadata sheet for a wheat leaf RNA virome experiment with 3 healthy leaves, 3 diseased leaves, 1 extraction blank, and 1 positive control, all Illumina PE150 on an RNA workflow. Then count how many biological samples fall in each treatment group.
One solution:

```
mkdir -p ~/viromics_course/phase2_experimental_design/exercise1
cd ~/viromics_course/phase2_experimental_design/exercise1

cat > plant_rna_virome_metadata.tsv << 'EOF'
sample_id   environment host    treatment   replicate   nucleic_acid    enrichment  control_t
Leaf_H_01   plant_leaf  wheat   healthy 1   RNA rRNA_depletion  biological_sample   batch1  l
Leaf_H_02   plant_leaf  wheat   healthy 2   RNA rRNA_depletion  biological_sample   batch1  l
Leaf_H_03   plant_leaf  wheat   healthy 3   RNA rRNA_depletion  biological_sample   batch1  l
Leaf_D_01   plant_leaf  wheat   diseased    1   RNA rRNA_depletion  biological_sample   batch
Leaf_D_02   plant_leaf  wheat   diseased    2   RNA rRNA_depletion  biological_sample   batch
Leaf_D_03   plant_leaf  wheat   diseased    3   RNA rRNA_depletion  biological_sample   batch
EXT_BLANK_01    blank   none    none    NA  RNA none    extraction_blank    batch1  lib1    l
POS_CTRL_01 mock    mock    mock    NA  RNA mock_community  positive_control    batch1  lib1
EOF

# Count biological samples per treatment group
awk -F'\t' 'NR>1 && $8=="biological_sample" {count[$4]++} END {for (t in count) print t, coun
  plant_rna_virome_metadata.tsv
```

Expected:

```
healthy 3
diseased 3
```

3.13 Key takeaways

- Experimental design matters more than the pipeline: a sharp, testable question — defining environment, host, comparison, and nucleic-acid type — should drive every downstream choice from sampling to analysis.
- Sample type, enrichment strategy, and the DNA-versus-RNA decision jointly determine which viruses you can detect; design separate DNA and RNA workflows rather than forcing one library to do both jobs.
- Every enrichment step (filtration, nuclease treatment, random amplification) biases the observed community, so the method must be reported clearly to make results interpretable and comparable.
- Negative and positive controls are part of the bioinformatics interpretation, not just wet-lab hygiene — especially for low-biomass samples where reagent contamination can dominate the reads.
- Biological replication (minimum three per group) answers whether a pattern is real, while a validated metadata sheet and FASTQ manifest keep samples traceable and the study reproducible (Minot et al. 2011).

3.14 Further reading

- Minot et al. (2011) demonstrates how sampling and enrichment choices shape an observed virome, motivating careful experimental design and controls.
- Chen et al. (2018) documents fastp, a fast all-in-one tool for the read quality control and adapter trimming that follows the sequencing choices planned here.
- The ICTV website provides the standard reference for naming and organizing the viruses your design aims to detect: https://ictv.global.
- The NCBI Sequence Read Archive hosts public raw reads useful for planning storage and benchmarking a design (NCBI 2026); browse it at https://www.ncbi.nlm.nih.gov/sra.

3.15 Chapter figure

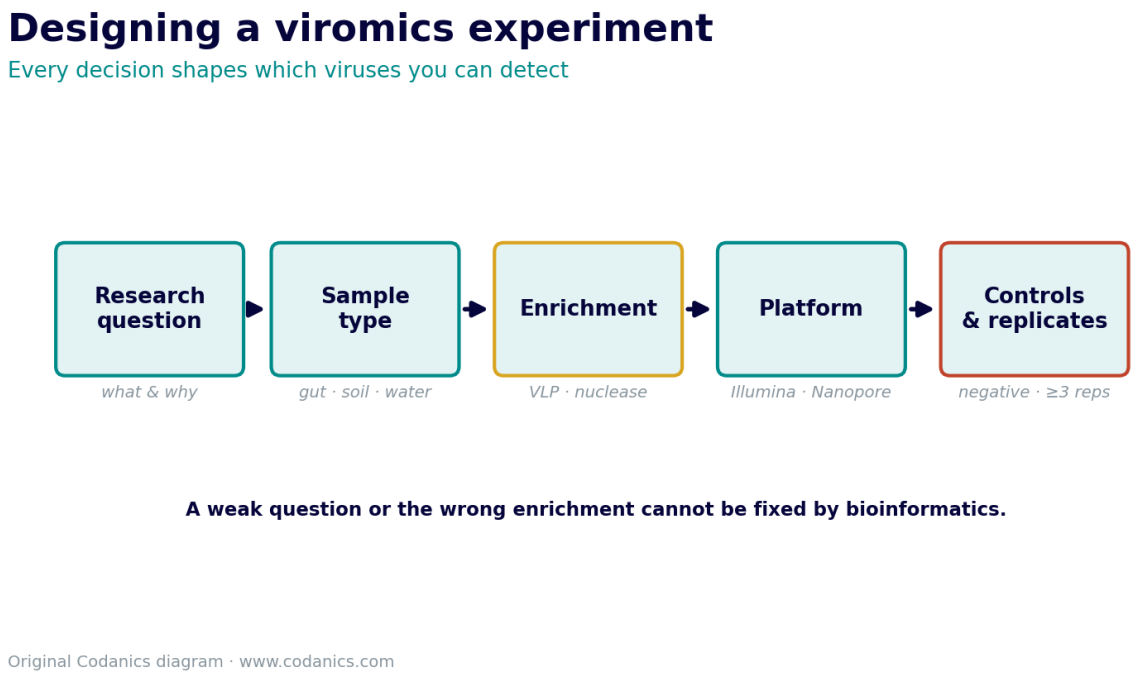


Figure 3.1: The viromics experimental-design workflow: a research question drives sample choice, enrichment, DNA or RNA extraction, controls, sequencing, and the bioinformatics pipeline.

Generate this figure

Save as: `images/ch02-experimental-design.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean educational infographic showing the viromics experimental-design workflow as a left-to-right pipeline. Begin on the left with a labelled "Research question" card, then flow through labelled stages: "Sample type" (with small icons for soil, gut, wastewater,

plant leaf, and respiratory swab), “Enrichment” (filter and nuclease icons), “DNA or RNA extraction” (two branching arrows, one DNA double helix, one RNA single strand), “Controls” (icons for extraction blank, library blank, and mock community), “Sequencing platform” (Illumina short-read and Nanopore long-read icons), and finally “Bioinformatics pipeline” shown as a Linux terminal leading to “Biological interpretation”. Add a subtle recurring fishing motif — a small net over the enrichment stage — to echo the fishing-in-the-ocean analogy. Use teal and navy Codanics branding, clear sans-serif labels, and connecting arrows between every stage.

3.16 Quiz: Experimental Design

Q1. Why are negative controls important in low-biomass viromics?

A. they remove the need for sequencing B. contamination can dominate low-biomass data C. they increase viral genome length D. they replace biological replicates

> **Note**
>
> **Answer:** B. Small contaminant signals can become a large fraction of the data when true biomass is low.

Q2. Which workflow requires reverse transcription?

A. DNA virome B. RNA virome C. protein annotation only D. Bowtie2 mapping

> **Note**
>
> **Answer:** B. RNA viromes are usually converted to cDNA before sequencing.

Q3. What is a disadvantage of viral enrichment?

A. it can introduce bias B. it prevents assembly C. it removes all viral reads D. it disables sequencing

> **Note**
>
> **Answer:** A. Filtration, nuclease treatment, and random amplification can change the observed community structure.

Q4. Which platform is usually practical for many high-accuracy paired-end samples?

A. Illumina B. Sanger only C. Western blot D. flow cytometry

> **Note**
>
> **Answer:** A. Illumina short reads are common for high-throughput community comparison.

Q5. What is the purpose of a FASTQ manifest?

A. link sample IDs to read files B. replace the reference database C. calculate Shannon diversity D. infer viral taxonomy

> **Note**
>
> **Answer:** A. A manifest prevents sample mix-ups and documents input files.

3.17 Interactive quiz: Experimental design

4 Linux Setup for Viromics on Ubuntu

Think of this chapter as preparing your **viromics laboratory bench**. In a wet lab you would never pour RNA extraction, DNA extraction, PCR, and sequencing reagents into one messy box — each protocol needs its own clean space and its own reagents. Bioinformatics is the same. Many viral tools depend on different versions of Python, R, HMMER, DIAMOND, or TensorFlow, and databases that assume specific layouts.

> **The rule for this book: one clean core environment, then a separate environment per complex viral tool.**

Installing everything into `base` is the fastest way to spend days untangling dependency conflicts. Bioconda recommends the `bioconda` and `conda-forge` channels with strict channel priority, and Miniforge is the lightweight, conda-forge-based installer that replaced Mambaforge for Ubuntu workstations.

Learning objectives — by the end of this chapter you will be able to:

- install Miniforge and configure Bioconda channels with strict priority;
- explain why viromics tools live in separate conda environments and design that layout;
- build a reproducible project folder structure and set the shared environment variables the book relies on;
- install and version-check the full viromics toolchain, from QC and assembly to host prediction; and
- set up the tool databases (VirSorter2, geNomad, CheckV, DRAM, eggNOG, iPHoP) and verify the whole install with one script.

4.1 Recommended computer resources

Viromics is disk- and memory-hungry, mostly because of the reference databases. For teaching and small practice datasets:

Component	Minimum	Better
RAM	16 GB	32–64 GB
CPU	4 cores	8–32 cores
Disk	200 GB	1–2 TB
OS	Ubuntu 22.04 / 24.04	Ubuntu server / HPC
Internet	Stable	Very stable

Full databases need much more disk. Some tools — **iPHoP**, **DRAM**, and large host databases — can each require hundreds of GB. On a student laptop you can demonstrate the commands; run them for real on a workstation, server, or HPC.

4.2 System preparation

Update the base system and install compilers and common utilities.

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y \
  build-essential wget curl git unzip zip tar gzip pigz tree htop \
  nano vim cmake make gcc g++ default-jre default-jdk
```

Check versions.

```
gcc --version
g++ --version
java -version
git --version
```

4.3 Install Miniforge

Skip this section only if you already have a working conda/mamba installation.

```
cd ~

wget https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-Linux-x86_64.
  -O Miniforge3-Linux-x86_64.sh

bash Miniforge3-Linux-x86_64.sh
source ~/.bashrc
```

During installation accept the license (`yes`), keep the default path, and answer `yes` to initialize conda. Then confirm and update the base tools.

```
conda --version
mamba --version
mamba update -n base -c conda-forge conda mamba -y
```

Configure channels with strict priority.

```
conda config --add channels bioconda
conda config --add channels conda-forge
conda config --set channel_priority strict
conda config --show channels
```

4.4 Project folder structure

```
mkdir -p ~/viromics_course
cd ~/viromics_course

mkdir -p raw_reads trimmed_reads qc_reports assemblies assembly_qc \
  viral_identification checkv votus taxonomy annotation abundance \
  host_prediction phylogeny diversity visualization metadata databases \
  logs scripts env_yamls code images

tree -L 1 ~/viromics_course
```

Set the shared paths and thread count once, in your shell profile, so every later command can reuse them.

```
cat >> ~/.bashrc << 'EOF'

# Viromics course paths
export VIROME=$HOME/viromics_course
export VIROME_DB=$HOME/viromics_course/databases
export THREADS=12
EOF

source ~/.bashrc
echo "$VIROME | $VIROME_DB | $THREADS"
```

4.5 Environment strategy

This looks like many environments, but it is professional practice: it stops one tool from breaking another and makes every step reproducible.

Environment	Purpose
`viromics-core`	QC, trimming, assembly, mapping, general utilities, R/Python visualization
`virsorter2`	Virus identification
`genomad`	Virus/plasmid identification and taxonomy
`checkv`	Viral genome quality and completeness
`virfinder`	R-based viral prediction
`deepvirfinder`	Deep-learning viral prediction
`vclust`	ANI-based viral clustering and vOTUs
`cd-hit`	Fast sequence clustering / dereplication
`vcontact2`	Gene-sharing network taxonomy
`vibrant`	Viral identification and functional annotation
`dramv`	DRAM-v viral functional annotation

Environment	Purpose
`eggnog`	eggNOG functional annotation
`iphop`	Host prediction
`phabox`	PhaGCN/PhaBOX taxonomy, host, and lifestyle tools
`wish`	WIsH host prediction from viral contigs

4.6 Fast path: create environments from files

If you would rather not type every `mamba create` command, ready-made `environment.yml` files are provided for the most important environments. Create each one with a single command (databases are still downloaded separately, as shown in the tool sections below).

```
conda env create -f envs/viromics-core.yml
conda env create -f envs/virsorter2.yml
conda env create -f envs/genomad.yml
conda env create -f envs/checkv.yml
conda env create -f envs/eggnog.yml
```

The sections below still explain each tool and its database, and cover the extra environments (VirFinder, DeepVirFinder, vClust, vConTACT2, VIBRANT, DRAM-v, iPHoP, PhaBOX, WIsH) that you install as you need them.

4.7 Core environment

This environment handles QC, trimming, assembly, mapping, plotting, and general utilities (Shen et al. 2016; Hyatt et al. 2010; Langmead and Salzberg 2012; Danecek et al. 2021).

```
mamba create -n viromics-core -y \
  -c conda-forge -c bioconda \
  python=3.11 r-base r-essentials r-tidyverse r-ggplot2 r-pheatmap \
  r-vegan r-ape bioconductor-phyloseq fastqc multiqc fastp trimmomatic \
  megahit spades quast sra-tools seqkit csvtk blast diamond hmmer \
  mmseqs2 prodigal cd-hit mafft iqtree bowtie2 samtools coverm bedtools \
  pandas numpy scipy matplotlib seaborn jupyterlab
```

Activate and check the key tools.

```
conda activate viromics-core
fastqc --version
multiqc --version
fastp --version
megahit --version
```

```
spades.py --version
quast.py --version
seqkit version
bowtie2 --version | head -n 1
samtools --version | head -n 1
coverm --version
R --version
python --version
```

Save a reproducible copy of the environment. Do this for every environment you build.

```
conda env export --no-builds > $VIROME/env_yamls/viromics-core.yml
```

4.8 Viral tool environments

4.8.1 VirSorter2

VirSorter2 detects diverse DNA and RNA viruses using a multi-classifier approach (Guo et al. 2021).

```
mamba create -n virsorter2 -y -c conda-forge -c bioconda virsorter=2
conda activate virsorter2
virsorter --version

mkdir -p $VIROME_DB/virsorter2
virsorter setup -d $VIROME_DB/virsorter2 -j $THREADS
conda env export --no-builds > $VIROME/env_yamls/virsorter2.yml
```

4.8.2 geNomad

geNomad identifies viruses and plasmids and provides taxonomy and annotation outputs (Camargo et al. 2024).

```
mamba create -n genomad -y -c conda-forge -c bioconda genomad
conda activate genomad
genomad --version

mkdir -p $VIROME_DB/genomad
genomad download-database $VIROME_DB/genomad
conda env export --no-builds > $VIROME/env_yamls/genomad.yml
```

4.8.3 CheckV

CheckV estimates viral contig quality, completeness, and contamination, and trims host flanks from proviruses (Nayfach et al. 2021). The database is downloaded separately, and its folder name varies by version, so locate it and export `CHECKVDB` into your profile.

```
mamba create -n checkv -y -c conda-forge -c bioconda checkv
conda activate checkv

mkdir -p $VIROME_DB/checkv
checkv download_database $VIROME_DB/checkv

CHECKV_PATH=$(find $VIROME_DB/checkv -maxdepth 2 -type d -name "checkv-db*" | head -n 1)
ln -sfn "$CHECKV_PATH" $VIROME_DB/checkv-db          # stable path reused across the book
echo 'export CHECKVDB=$VIROME_DB/checkv-db' >> ~/.bashrc
source ~/.bashrc
echo "$CHECKVDB"
conda env export --no-builds > $VIROME/env_yamls/checkv.yml
```

4.8.4 VirFinder

VirFinder is an R package that scores contigs as viral using k-mer signatures learned from reference genomes (Ren et al. 2017).

```
mamba create -n virfinder -y -c conda-forge -c bioconda \
  r-base r-virfinder r-tidyverse
conda activate virfinder

# Load the package and print its version to confirm the install
R -q -e "library(VirFinder); packageVersion('VirFinder')"
conda env export --no-builds > $VIROME/env_yamls/virfinder.yml
```

4.8.5 DeepVirFinder

DeepVirFinder predicts viral sequences with a deep-learning model and was designed to work well on short contigs (Ren et al. 2020). It is older and dependency-sensitive, so keep it isolated. Two routes are shown; use the source route when the conda package fails to solve.

Route A — conda package from the HCC channel:

```
mamba create -n deepvirfinder -y \
  -c hcc -c conda-forge -c bioconda deepvirfinder
conda activate deepvirfinder
dvf.py -h | head
```

Route B — GitHub source install:

```
mamba create -n deepvirfinder-src -y -c conda-forge -c bioconda \
  python=3.6 numpy theano keras scikit-learn biopython
conda activate deepvirfinder-src

mkdir -p $VIROME/software && cd $VIROME/software
git clone https://github.com/jessieren/DeepVirFinder.git
cd DeepVirFinder
python dvf.py -h
conda env export --no-builds > $VIROME/env_yamls/deepvirfinder.yml
```

> **What we teach as the core method**
>
> For modern pipelines, treat **VirSorter2 + geNomad + CheckV** as the core viral-identification stack, then use VirFinder and DeepVirFinder as additional evidence layers rather than standalone verdicts.

4.8.6 vClust and CD-HIT

vClust calculates average nucleotide identity (ANI) and clusters viral genomes into vOTUs, typically at a 95% ANI threshold (Zielezinski et al. 2025). CD-HIT is already in `viromics-core`, but a dedicated environment keeps its version stable for teaching and dereplication (Fu et al. 2012).

```
mamba create -n vclust -y -c conda-forge -c bioconda vclust
conda activate vclust
vclust --help | head
conda env export --no-builds > $VIROME/env_yamls/vclust.yml

mamba create -n cd-hit -y -c conda-forge -c bioconda cd-hit
conda activate cd-hit
cd-hit -h | head
cd-hit-est -h | head
conda env export --no-builds > $VIROME/env_yamls/cd-hit.yml
```

4.8.7 vConTACT2

vConTACT2 classifies prokaryotic viruses using gene-sharing networks; the official install pairs it with `mcl`, `blast`, and `diamond` (Bin Jang et al. 2019).

```
mamba create -n vcontact2 -y -c conda-forge -c bioconda \
  python=3 vcontact2 mcl blast diamond prodigal
conda activate vcontact2
vcontact2 --help | head
conda env export --no-builds > $VIROME/env_yamls/vcontact2.yml
```

4.8.8 VIBRANT

VIBRANT recovers and annotates bacterial and archaeal viruses from metagenomic assemblies (Kieft, Zhou, and Anantharaman 2020). Its database is prepared with `download-db.sh`.

```
mamba create -n vibrant -y -c conda-forge -c bioconda vibrant
conda activate vibrant
VIBRANT_run.py -h | head

mkdir -p $VIROME_DB/vibrant
cd $VIROME_DB/vibrant
download-db.sh
conda env export --no-builds > $VIROME/env_yamls/vibrant.yml
```

4.8.9 DRAM / DRAM-v

DRAM annotates microbial and viral genomes; DRAM-v is especially useful for viral functional annotation and auxiliary metabolic gene interpretation (Shaffer et al. 2020). DRAM ships its own `environment.yaml`, and databases are prepared with `DRAM-setup.py`.

```
mkdir -p $VIROME/software && cd $VIROME/software
wget https://raw.githubusercontent.com/WrightonLabCSU/DRAM/master/environment.yaml \
  -O DRAM_environment.yaml

mamba env create -f DRAM_environment.yaml -n dramv
conda activate dramv
DRAM-setup.py --help | head
```

Prepare the databases. On a workstation, add `--skip_uniref`: UniRef90 is enormous and pushes memory far beyond a typical machine, and DRAM works well for viral annotation without it. Then confirm the configuration.

```
mkdir -p $VIROME_DB/DRAM_data

DRAM-setup.py prepare_databases \
  --output_dir $VIROME_DB/DRAM_data \
  --skip_uniref \
  --threads $THREADS \
  --verbose

DRAM-setup.py print_config
conda env export --no-builds > $VIROME/env_yamls/dramv.yml
```

> **DRAM database setup is heavy**
>
> Even with `--skip_uniref`, DRAM database preparation is one of the most resource-intensive steps in this book. On student laptops, demonstrate the commands; run them for real on a workstation, server, or HPC.

4.8.10 eggNOG-mapper

eggNOG-mapper annotates predicted proteins by orthology (Cantalapiedra et al. 2021). Define `EGGNOG_DATA_DIR` before downloading, and persist it to your profile so every run finds the same database.

```
mamba create -n eggnog -y -c conda-forge -c bioconda eggnog-mapper
conda activate eggnog
emapper.py --version

mkdir -p $VIROME_DB/eggnog
export EGGNOG_DATA_DIR=$VIROME_DB/eggnog
download_eggnog_data.py --data_dir $EGGNOG_DATA_DIR   # answer y when prompted

echo "export EGGNOG_DATA_DIR=$VIROME_DB/eggnog" >> ~/.bashrc
source ~/.bashrc
conda env export --no-builds > $VIROME/env_yamls/eggnog.yml
```

4.8.11 iPHoP

iPHoP predicts the host genus of uncultivated bacteriophages and archaeal viruses by integrating several host-prediction signals. It is available through Bioconda; the database is large.

```
mamba create -n iphop -y -c conda-forge -c bioconda iphop
conda activate iphop
iphop --help | head

mkdir -p $VIROME_DB/iphop
iphop download --db_dir $VIROME_DB/iphop

du -sh $VIROME_DB/iphop
conda env export --no-builds > $VIROME/env_yamls/iphop.yml
```

> **The iPHoP database is very large**
>
> The iPHoP host database runs to hundreds of GB. Confirm you have the disk space before downloading, and prefer server or HPC storage when teaching with real data.

4.8.12 PhaBOX / PhaGCN

PhaGCN moved into PhaBOX 2, which now covers taxonomy classification, host prediction, lifestyle prediction, vOTU grouping, tree building, and viral protein annotation (Shang et al. 2026). Install from Bioconda first; fall back to source if the package is unavailable or outdated, then download the database.

```
mamba create -n phabox -y -c conda-forge -c bioconda phabox
conda activate phabox
phabox2 --help
```

Source install (only if the Bioconda package fails):

```
mkdir -p $VIROME/software && cd $VIROME/software
git clone https://github.com/KennthShang/PhaBOX.git
cd PhaBOX
python -m pip install -r requirements.txt
python -m pip install .
```

Download the PhaBOX database:

```
mkdir -p $VIROME_DB/phabox
cd $VIROME_DB/phabox
wget https://github.com/KennthShang/PhaBOX/releases/download/v2/phabox_db_v2_2.zip
unzip phabox_db_v2_2.zip
conda env export --no-builds > $VIROME/env_yamls/phabox.yml
```

4.8.13 WIsH

WIsH predicts prokaryotic hosts of phage contigs using Markov models and is compiled from source (Galiez et al. 2017). Build it inside a small environment that provides the compilers, then add the binary to your `PATH`.

```
mamba create -n wish -y -c conda-forge -c bioconda \
  cmake make gcc_linux-64 gxx_linux-64
conda activate wish

mkdir -p $VIROME/software && cd $VIROME/software
git clone https://github.com/soedinglab/WIsH.git
cd WIsH
cmake .
make

echo "export PATH=$VIROME/software/WIsH:\$PATH" >> ~/.bashrc
source ~/.bashrc
WIsH -h | head
conda env export --no-builds > $VIROME/env_yamls/wish.yml
```

4.8.14 CRISPR spacer tools

CRISPR-based host prediction needs only BLAST+, `seqkit`, `bedtools`, and `samtools` — all already in `viromics-core`. Bacterial and archaeal genomes record previous phage encounters as CRISPR spacers, so a high-identity BLAST match between a host spacer and a viral contig is evidence of a host relationship. In the complete pipeline chapter you will build a spacer database with `makeblastdb` and run a short-word `blastn` search against your viral contigs.

```
conda activate viromics-core
blastn -version
makeblastdb -version
seqkit version
bedtools --version
```

4.9 Tool map for the complete pipeline

Step	Tool	Main input	Main output
Raw data download	SRA Toolkit	SRA accession	FASTQ files
QC	FastQC / MultiQC	FASTQ	HTML reports
Trimming	fastp / Trimmomatic	Raw FASTQ	Clean FASTQ
Assembly	MEGAHIT / metaSPAdes	Clean FASTQ	contigs FASTA
Assembly QC	QUAST	contigs FASTA	assembly report
Virus detection	VirSorter2 / geNomad / VirFinder / DeepVirFinder	contigs FASTA	viral candidates
Quality	CheckV	viral contigs FASTA	quality/completeness table
Dereplication	vClust / CD-HIT	viral FASTA	vOTU clusters
Taxonomy	vConTACT2 / geNomad / PhaBOX	viral FASTA / proteins	taxonomy table
Gene calling	Prodigal	viral FASTA	proteins / GFF
Annotation	DRAM-v / VIBRANT / eggNOG	viral FASTA / proteins	functional tables
Abundance	Bowtie2 / CoverM	reads + viral contigs	coverage table
Host prediction	iPHoP / WIsH / CRISPR BLAST	viral FASTA + host DB	host predictions
Phylogeny	MAFFT / IQ-TREE	marker genes	alignment / tree
Visualization	R / Python	tables	publication figures

4.10 Installation check

Rather than checking each environment by hand, run one script that reports every environment, core and viral tool versions, and database sizes. The book ships a ready-made version.

The core of that script is shown below; download the full copy above rather than retyping it.

```
cat > $VIROME/scripts/check_viromics_installation.sh << 'EOF'
#!/usr/bin/env bash
set -e

echo "Conda environments"
conda env list

echo "Core tools"
conda run -n viromics-core fastqc --version
conda run -n viromics-core fastp --version
conda run -n viromics-core megahit --version
conda run -n viromics-core spades.py --version
conda run -n viromics-core quast.py --version
conda run -n viromics-core seqkit version
conda run -n viromics-core bowtie2 --version | head -n 1
conda run -n viromics-core samtools --version | head -n 1
conda run -n viromics-core coverm --version

echo "Viral tools"
conda run -n virsorter2 virsorter --version || true
conda run -n genomad genomad --version || true
conda run -n checkv checkv -h | head -n 2 || true
conda run -n virfinder R -q -e "packageVersion('VirFinder')" || true
conda run -n vclust vclust --help | head -n 2 || true
conda run -n vcontact2 vcontact2 --help | head -n 2 || true
conda run -n vibrant VIBRANT_run.py -h | head -n 2 || true
conda run -n dramv DRAM.py --help | head -n 2 || true
conda run -n eggnog emapper.py --version || true
conda run -n iphop iphop --help | head -n 2 || true

echo "Database sizes"
du -sh $VIROME_DB/* 2>/dev/null || true
EOF

chmod +x $VIROME/scripts/check_viromics_installation.sh
bash $VIROME/scripts/check_viromics_installation.sh
```

Estimated resources on 12 threads and 32 GB RAM: the core environment usually installs in 20–60 minutes and 10–25 GB. Databases range from ~50 GB to more than 500 GB depending on the tools selected. CheckV and geNomad are moderate; DRAM and iPHoP are heavy.

4.10.1 Environment cheat sheet

Keep one short reference for which environment to activate for each task. A ready-made copy is included with the book.

```
Core tools:              conda activate viromics-core
Virus identification:    conda activate virsorter2 | genomad | virfinder | deepvirfinder
Quality:                 conda activate checkv
Clustering:              conda activate vclust | cd-hit
Taxonomy:                conda activate vcontact2 | phabox | genomad
Functional annotation:   conda activate vibrant | dramv | eggnog
Host prediction:         conda activate iphop | wish
Main course folder:      cd $VIROME
Database folder:         cd $VIROME_DB
```

⚠ Common mistakes

- **Installing into `base`.** Never install viromics tools into the base conda environment. Always `mamba create -n toolname ...` then `conda activate toolname`.
- **Forgetting the database.** A tool without its database fails at runtime, not at install: VirSorter2 with no setup database, CheckV with `CHECKVDB` unset, geNomad with no `genomad_db`, VIBRANT with no `download-db.sh` run.
- **Mixing old and new tools in one environment.** DeepVirFinder, VirFinder, DRAM, VirSorter2, and PhaBOX have conflicting dependencies. Keep them separate so one upgrade cannot silently break another.

ℹ Exercise 1: build a `virome-test` environment

Create a throwaway environment called `virome-test` containing `fastqc`, `fastp`, `seqkit`, and `multiqc`, confirm each tool runs, then export the environment file.
One solution:

```
mamba create -n virome-test -y -c conda-forge -c bioconda \
  fastqc fastp seqkit multiqc

conda activate virome-test
fastqc --version
fastp --version
seqkit version
multiqc --version

conda env export --no-builds > $VIROME/env_yamls/virome-test.yml
```

ℹ Exercise 2: a `check_fastq_pairs.sh` guard script

Write a small script that verifies both paired-end FASTQ files exist before an analysis starts, and prints a clear message if either is missing.
One solution:

```
cat > $VIROME/scripts/check_fastq_pairs.sh << 'EOF'
#!/usr/bin/env bash
R1=$VIROME/raw_reads/samples_R1.fastq.gz
R2=$VIROME/raw_reads/samples_R2.fastq.gz

if [[ -f "$R1" && -f "$R2" ]]; then
  echo "FASTQ pair found:"
  echo "$R1"
  echo "$R2"
else
  echo "Missing FASTQ file. Check the raw_reads folder."
fi
EOF

chmod +x $VIROME/scripts/check_fastq_pairs.sh
bash $VIROME/scripts/check_fastq_pairs.sh
```

Guard scripts like this are worth writing once and reusing: they turn a confusing mid-pipeline crash into a one-line, human-readable error.

4.11 Key takeaways

- Keep one clean `viromics-core` environment for QC, assembly, and mapping, and give every complex viral tool its own isolated conda environment so incompatible dependencies never collide.
- Install with Miniforge and mamba, enabling the `conda-forge` and `bioconda` channels with strict channel priority for fast, reproducible solves.
- A tool is only half-installed until its database is in place: VirSorter2 (Guo et al. 2021), geNomad (Camargo et al. 2024), CheckV (Nayfach et al. 2021), eggNOG-mapper (Cantalapiedra et al. 2021), and iPHoP (Roux et al. 2023) each need a separate download, and iPHoP and DRAM databases can each reach hundreds of GB.
- Store all databases in one central `$VIROME_DB` folder and persist paths such as `CHECKVDB` and `EGGNOG_DATA_DIR` in your shell profile so every step finds them.
- Export each environment with `conda env export --no-builds` and verify the whole stack with a single installation-check script before running any real analysis.

4.12 Further reading

- geNomad official documentation and GitHub for install, database download, and end-to-end usage (Camargo et al. 2024); project site: https://github.com/apcamargo/genomad.
- VirSorter2 for the multi-classifier setup and `virsorter setup` database step (Guo et al. 2021).
- CheckV for how the quality database is built and consumed by the pipeline (Nayfach et al. 2021).

- Bioconda channel configuration and package discovery, the backbone of every environment here (Shen et al. 2016).

4.13 Chapter figure

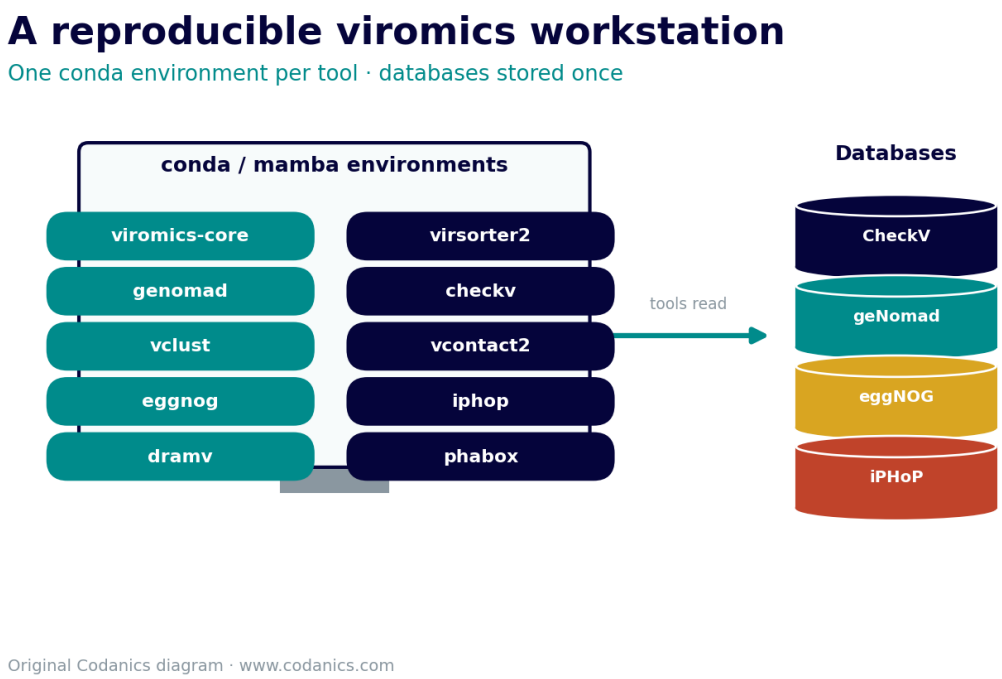


Figure 4.1: A Linux viromics workstation: Ubuntu and Miniforge at the base, a clean core environment beside isolated per-tool environments, and a central databases folder feeding the analysis pipeline.

Generate this figure

Save as: `images/ch03-workstation-setup.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean educational infographic of a Linux bioinformatics workstation set up for viromics. At the bottom, show a foundation layer labelled "Ubuntu" with a "Miniforge / mamba" installer and three stacked channel labels "conda-forge", "bioconda", "strict priority". Above it, draw a large clean box labelled "viromics-core (QC · assembly · mapping · R/Python)" next to a neat row of smaller isolated boxes labelled "virsorter2", "genomad", "checkv", "vclust", "vcontact2", "dramv", "eggnog", "iphop", "phabox", "wish", each drawn as a separate lab-bench station to emphasise isolation. On the right, show a folder tree icon for the project directory with sub-folders raw_reads, assemblies, viral_identification, checkv, taxonomy, host_prediction, visualization, and a highlighted central "databases" cylinder connected by arrows to several tool boxes. Add a small terminal window showing a green check mark labelled "installation check". Use teal and navy Codanics branding, clear sans-serif labels,

and a tidy left-to-right, bottom-to-top flow.

4.14 Quiz: Linux Setup

Q1. Why should tools be installed in separate environments?

A. to avoid dependency conflicts B. to increase read length C. to remove viruses D. to replace databases

Note

Answer: A. Many viral tools require different dependency versions.

Q2. Which database variable is commonly set for CheckV?

A. CHECKVDB B. FASTQDB C. TREEPATH D. PHAGEDIR

Note

Answer: A. CheckV needs a database path.

Q3. Which command exports a conda environment?

A. conda env export –no-builds B. conda delete all C. mamba tree D. fastqc export

Note

Answer: A. Environment export improves reproducibility.

Q4. Which environment should run FastQC and MEGAHIT in this book?

A. viromics-core B. checkv C. iphop D. wish

Note

Answer: A. The core environment contains general pipeline tools.

Q5. Why are databases stored in a central folder?

A. to document and reuse large data B. to delete outputs C. to trim reads D. to make the CPU faster

Note

Answer: A. Database paths must be documented for reproducibility.

4.15 Interactive quiz: Linux setup

5 ViroProfiler: an automated A–Z pipeline

5.1 ViroProfiler: an automated A–Z viromics pipeline

Before you build a viromics workflow by hand, it helps to see the whole thing run end to end. **ViroProfiler** is a containerized Nextflow pipeline that turns raw reads into an annotated, quantified catalogue of viral genomes — read QC, assembly, viral identification, CheckV quality, vOTU clustering, taxonomy, host and lifestyle prediction, functional annotation, and abundance — from a single command. This chapter is a tested, reproducible walkthrough: install it, set up and **verify** its databases, run the bundled test dataset, then run a full analysis on public virome reads — including the failure modes the pipeline does not warn you about, and how to interpret every output it produces.

Run this chapter first to get real results quickly and see the shape of a complete viromics study. Then work through the step-by-step pipeline to understand what each stage is doing under the hood and when to deviate from the defaults.

Learning objectives — by the end of this chapter you will be able to:

- install ViroProfiler with Singularity/Apptainer and a pinned Nextflow, in a self-contained project;
- set up the databases and, crucially, **verify** they are complete rather than trusting a "success" message;
- run the bundled test dataset and a full analysis on public virome reads, end to end;
- recognize and fix the pipeline's real failure modes (read-only `$HOME`, dead dbCAN downloads, silent 1-CPU/10-GB defaults, iPHoP out-of-memory); and
- interpret every output table — vOTUs, CheckV quality, abundance, taxonomy, host, lifestyle, and function.

> Two pipelines, one goal
>
> This chapter runs an **automated** pipeline: one command orchestrating many tools. The complete pipeline chapter then rebuilds the same workflow **by hand**, tool by tool, so you understand and control every step. Read this one for the whole picture and fast results; read that one to learn the internals.

> Warning
>
> This is a heavyweight, real-world pipeline: it needs a Linux workstation, Singularity/Apptainer, and roughly **420 GB of databases**. The `data/toy/` and `data/example/` datasets used elsewhere in the book are for practicing commands; ViroProfiler is where you run a genuine analysis on real virome data.

5.2 What ViroProfiler does

ViroProfiler takes raw metagenomic (or virome) sequencing reads and produces an annotated, quantified catalogue of viral genomes. The published description is:

> Ru, Jinlong, et al. "ViroProfiler: a containerized bioinformatics pipeline for viral metagenomic data analysis." *Gut Microbes* 15.1 (2023): 2192522. https://doi.org/10.1080/19490976.2023.2192522

5.2.1 The workflow

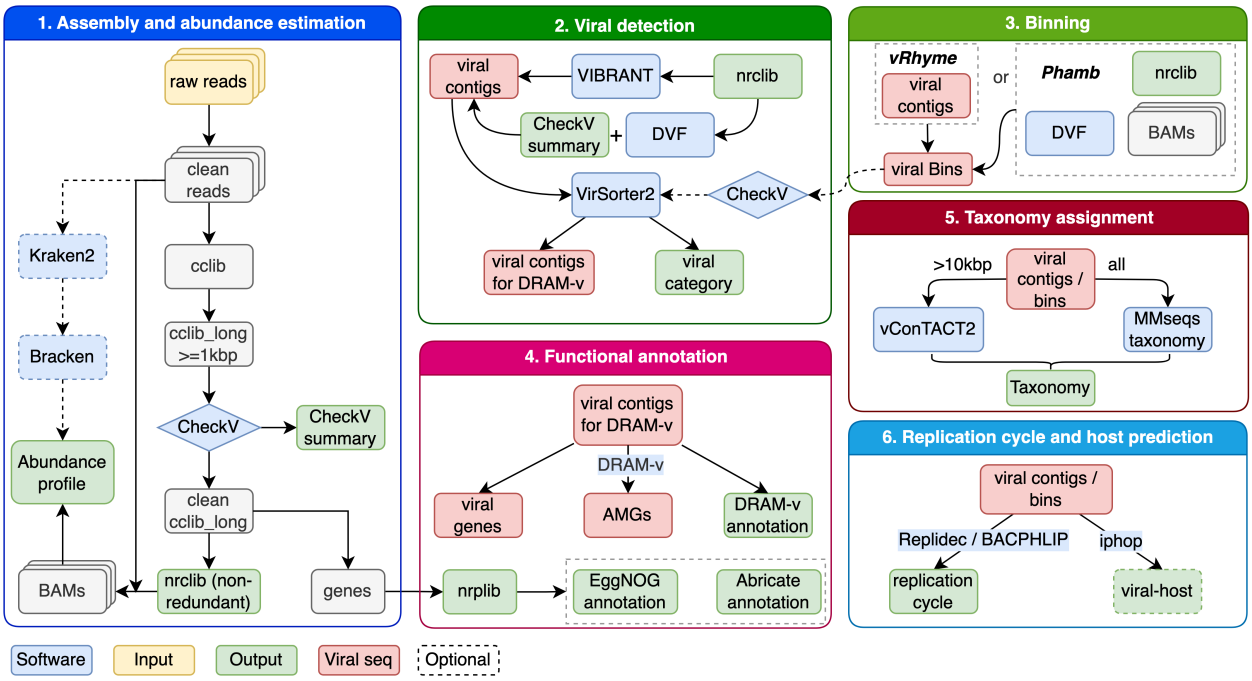


Figure 5.1: The ViroProfiler workflow, from the project repository.

The stages, shown in Figure 5.1, are:

1. **Read QC**: quality control and adapter/quality trimming
2. **Assembly**: metagenomic assembly into contigs
3. **Viral identification**: find which contigs are viral
4. **Quality assessment**: completeness and contamination of viral genomes
5. **Clustering**: collapse to a non-redundant viral contig library (vOTUs)
6. **Gene prediction and annotation**: genes, function, AMGs
7. **Taxonomy**: classify viral contigs
8. **Host prediction**: predict bacterial hosts for phages
9. **Lifestyle prediction**: lytic vs. lysogenic
10. **Abundance**: map reads back to quantify each vOTU per sample

5.2.2 Tools used

Tool	Role	Links
Nextflow	Workflow engine	docs · paper
nf-core	Pipeline framework	paper
FastQC	Read quality reports	docs
fastp	Trimming and filtering	paper
BBMap	Decontamination	guide
metaSPAdes	Metagenomic assembly	paper
Bowtie 2	Read mapping	paper
CoverM	Coverage and abundance	:
CheckV	Viral genome QC	paper
VirSorter2	Viral contig detection	paper
DeepVirFinder	ML viral detection	paper
VIBRANT	Viral detection + annotation	paper
vRhyme	Viral binning	paper
Phamb	Viral binning	paper
DRAM-v	Gene annotation, AMGs	paper
dbCAN	CAZyme annotation	paper · S3 releases
eggNOG-mapper	Orthology annotation	paper
abricate	AMR/virulence genes	:
vConTACT2	Gene-sharing taxonomy	paper
MMseqs2	Fast search / taxonomy	paper
Kraken2 / Bracken	Read-level taxonomy	paper
iPHoP	Host prediction	paper
BACPHLIP	Lifestyle prediction	paper
Replidec	Lifestyle prediction	:

5.3 System requirements

Resource	Minimum	Recommended	Notes
OS	Linux x86-64	:	Tested on Linux Mint 22 / Ubuntu noble
CPU	4 cores	12+	vConTACT2 and DRAM benefit most
RAM	24 GB	32 GB+	iPHoP peaked at **33.7 GB** in the reference run
Swap	32 GB	32 GB	**Required** at 32 GB RAM: iPHoP exceeds physical RAM, see Section 5.4.6

Resource	Minimum	Recommended	Notes
Disk	450 GB	600 GB+	Databases alone are ~420 GB
Filesystem	ext4/xfs	:	**Not exFAT/NTFS**: see note below

Filesystem matters. Singularity's layer cache names entries `sha256:<hex>` and its build sandbox needs symlinks. exFAT supports neither, and image builds fail with `could not finalize cached file ... invalid argument`. Use ext4 or xfs.

Disk budget (measured on a complete install):

Database	Size
iPHoP	199 GB
DRAM	203 GB
VIBRANT	11 GB
VirSorter2	11 GB
CheckV	7 GB
taxonomy (MMseqs vRefSeq)	4 GB
Container images	~5 GB
Total	**~440 GB**

5.4 Installation

Everything below is self contained. You can copy each block in turn and end up with a working installation without needing any file from this repository.

All paths derive from a single variable, `VP_HOME`, so nothing is tied to one machine.

Every command in this chapter is also packaged as a ready-to-run script — download the set and run them in order, or copy the blocks by hand:

5.4.1 Prerequisites

Singularity (or Apptainer) and conda must be present.

```
# Debian / Ubuntu / Linux Mint
sudo apt update && sudo apt install -y singularity-container

# Fedora / RHEL
sudo dnf install -y singularity-ce

singularity --version
```

If conda is missing, install Miniforge. Verify both:

```
singularity --version     # expect 3.x or 4.x
conda --version
```

5.4.2 Create the project folder

Pick a disk with at least **650 GB free**. Databases take about 420 GB, and the work directory needs room for intermediate files.

```
# Choose any location you like. Everything the pipeline writes stays inside it.
mkdir -p ~/viroprofiler
cd ~/viroprofiler

export VP_HOME="$(pwd)"
echo "VP_HOME is ${VP_HOME}"

# Check you really have the space
df -h "${VP_HOME}"
```

Create the directory layout:

```
mkdir -p "${VP_HOME}"/{db,work,data,container-home,nextflow-singularity-cache,singularity-cache
ls "${VP_HOME}"
```

Directory	Purpose
`db/`	Databases, about 420 GB
`work/`	Nextflow intermediate files
`data/`	Your input reads
`container-home/`	Writable HOME for containers, see Section 5.8.1
`nextflow-singularity-cache/`	Built `.img` files
`singularity-cache/`	OCI layer cache
`singularity-tmp/`	Image build sandbox

5.4.3 Add the Singularity settings to your shell

These three variables keep every cache inside the project. Without them, Singularity writes multi gigabyte images into `$HOME`, which is small on most systems, and image builds fail when it fills up.

```
cat >> ~/.bashrc <<EOF

# ---------------- ViroProfiler ----------------
export VP_HOME="${VP_HOME}"
```

```
export NXF_SINGULARITY_CACHEDIR="${VP_HOME}/nextflow-singularity-cache"
export SINGULARITY_CACHEDIR="${VP_HOME}/singularity-cache"
export SINGULARITY_TMPDIR="${VP_HOME}/singularity-tmp"
export NXF_VER=25.10.2
# ----------------------------------------------
EOF

source ~/.bashrc
```

Confirm they are set:

```
echo "${VP_HOME}"
echo "${NXF_SINGULARITY_CACHEDIR}"
echo "${SINGULARITY_CACHEDIR}"
echo "${SINGULARITY_TMPDIR}"
```

> **Warning**
>
> **The filesystem matters.** Use ext4 or xfs. Singularity's layer cache names entries `sha256:<hex>` and its build sandbox needs symlinks, neither of which exFAT or NTFS can represent. Image builds fail there with `could not finalize cached file ... invalid argument`.
> **If you re-install later**, edit these lines rather than appending new ones. A stale `NXF_SINGULARITY_CACHEDIR` pointing at an old project is a common cause of `Failed to create Singularity cache directory`.

5.4.4 Conda environment and Nextflow

```
conda create -n viroprofiler_env -c bioconda -c conda-forge nextflow -y
conda activate viroprofiler_env

# Pin the version, see the warning below
export NXF_VER=25.10.2
nextflow -v
```

Expected output:

```
nextflow version 25.10.2.10555
```

> **Caution**
>
> **Nextflow 26.x cannot run this pipeline.** Its stricter config parser rejects ViroProfiler v0.2.4's own `nextflow.config`, specifically the legacy `def check_max(obj, type) {...}`

function, and the run dies before any process starts with `Unexpected input: '('`. Always keep `NXF_VER=25.10.2` exported.

If `nextflow -v` reports a different version, another copy is ahead on your `PATH`. Force the environment's own:

```
export PATH="${CONDA_PREFIX}/bin:${PATH}"
nextflow -v
```

5.4.5 Create `local.config`

This file carries every fix described in section 6. It detects your CPU and RAM and assigns resources to every process explicitly, which the pipeline itself does not do.

Create it in one command:

Show the full local.config to create (click to expand)

```
cat > "${VP_HOME}/local.config" <<'CONFIGEOF'
/*
 * ViroProfiler local execution config.
 * Paths come from $VP_HOME, or from the directory you launch nextflow in.
 * Resources are detected from the machine and divided into tiers.
 */

def VP_HOME = System.getenv('VP_HOME') ?: System.getProperty('user.dir')

def detectedCpus = Runtime.runtime.availableProcessors()
def detectedMemGb = {
    try {
        def line = new File('/proc/meminfo').readLines().find { it.startsWith('MemTotal') }
        return (long) ((line.replaceAll(/\D/, '') as long) / 1024 / 1024)
    } catch (Exception e) { return 16L }
}()

// Leave 1 core and 3 GB for the OS. A fully committed workstation becomes
// unusable, and at 100% commitment an OOM kills tasks instead of slowing them.
def MAX_CPUS   = (System.getenv('VP_MAX_CPUS')   ?: "${Math.max(1, detectedCpus - 1)}") as in
def MAX_MEM_GB = (System.getenv('VP_MAX_MEM_GB') ?: "${Math.max(8, detectedMemGb - 3)}") as i

def CPU_HEAVY  = MAX_CPUS
def CPU_MEDIUM = Math.max(2, (int) (MAX_CPUS / 2))
def CPU_LIGHT  = Math.max(2, (int) (MAX_CPUS / 6))
def MEM_HEAVY  = "${MAX_MEM_GB}.GB"
def MEM_MEDIUM = "${Math.max(6, (int) (MAX_MEM_GB / 2))}.GB"
def MEM_LIGHT  = "${Math.max(4, (int) (MAX_MEM_GB / 6))}.GB"
def CPU_IPHOP  = Math.max(1, Math.min(8, MAX_CPUS - 2))

singularity {
    enabled    = true
    autoMounts = true
    cacheDir   = "${VP_HOME}/nextflow-singularity-cache"

    // -B binds the project so containers can read db/ and write work/.
    // Never bind over $HOME: Nextflow separately binds the pipeline's scripts
    // from ~/.nextflow/assets/.../bin onto PATH, and mounting over your home
    // hides that bind, giving "run_checkv.sh: command not found".
    //
    // --writable-tmpfs gives an in-memory overlay so DRAM can create the
    // symlink /opt/conda/db2 inside the read-only image.
    runOptions = "-B ${VP_HOME} --writable-tmpfs"
}

// Writable HOME for containers. Nextflow uses --no-home, so $HOME points into
// the read-only image layer and VirSorter2 dies creating ~/.virsorter.
env {
    HOME = "${VP_HOME}/container-home"
}
```

Check that it parses and see what it allocated on your machine:

```
cd "${VP_HOME}"
nextflow config . -c local.config 2>&1 | head -5
```

You should see a line such as:

```
[local.config] detected 12 CPUs / 31 GB RAM
[local.config] using    11 CPUs / 28 GB
```

5.4.6 Swap

Required if you have 32 GB RAM or less. In the reference run iPHoP peaked at **33.7 GB**, which is more than the physical RAM of the machine it ran on. It completed only because swap absorbed the overshoot. Without swap the kernel kills it.

```
# Put the swapfile on a disk with room to spare
SWAPFILE=/swapfile           # or ${VP_HOME}/swapfile on a large data disk

sudo fallocate -l 32G "${SWAPFILE}" || \
  sudo dd if=/dev/zero of="${SWAPFILE}" bs=1M count=32768 status=progress
sudo chmod 600 "${SWAPFILE}"
sudo mkswap "${SWAPFILE}"
sudo swapon "${SWAPFILE}"

# Make it permanent
echo "${SWAPFILE} none swap sw 0 0" | sudo tee -a /etc/fstab

# Keep swap as an emergency reserve rather than routine paging
sudo sysctl -w vm.swappiness=10
echo "vm.swappiness=10" | sudo tee /etc/sysctl.d/99-swappiness.conf

swapon --show
free -h
```

> **!** Important
>
> A `memory` directive in Nextflow governs Nextflow's own **scheduling**. It cannot reserve RAM from the OS and it cannot create 33.7 GB on a 31 GB machine. Swap is the only thing that turns "killed" into "completes".

5.4.7 Fetch the pipeline

```
conda activate viroprofiler_env
export NXF_VER=25.10.2

nextflow pull deng-lab/viroprofiler -r main
nextflow info deng-lab/viroprofiler
```

5.4.8 Installation checklist

Do not continue past a failing check.

Check	Command	Expected
Singularity	`singularity --version`	a version prints
Nextflow	`nextflow -v`	**25.10.2**
Variables	`echo $VP_HOME $NXF_SINGULARITY_CACHEDIR`	both set, inside your project
Swap	`swapon --show`	your swapfile is listed
Space	`df -h "$VP_HOME"`	650 GB or more free
Pipeline	`nextflow info deng-lab/viroprofiler`	revision info prints

5.5 Databases

The full set is about **420 GB** and takes several hours. Two of them, iPHoP and DRAM, account for most of it.

Database	Size	Used by
iPHoP	199 GB	Host prediction
DRAM	203 GB	Gene annotation, AMGs
VIBRANT	11 GB	Viral identification
VirSorter2	11 GB	Viral identification
CheckV	7 GB	Genome quality
taxonomy (MMseqs vRefSeq)	4 GB	Taxonomy

5.5.1 Run the pipeline's setup mode

```
cd "${VP_HOME}"
conda activate viroprofiler_env
export NXF_VER=25.10.2
```

```
nextflow run deng-lab/viroprofiler \
    -r main -profile singularity \
    -c "${VP_HOME}/local.config" \
    -work-dir "${VP_HOME}/work" \
    --mode setup \
    --db "${VP_HOME}/db"
```

> ⚠ Warning
>
> **Do not use `-resume` for database setup.** The `DB_*` processes declare no output files, so a cached "success" is replayed even after you delete the directory it was supposed to fill. Without `-resume` each task re-runs but short circuits in seconds when its directory is already complete.

This step will finish with an error message about DRAM. **That is expected**, and Section 5.5.2 repairs it. The reason is in Section 5.8.3: DRAM downloads three dbCAN files from a host that now returns an HTML page for every URL, so its setup aborts partway.

5.5.2 Complete the DRAM database

DRAM's own setup stops before it builds the description database, and before it processes the viral, peptidase and VOGdb files it already downloaded. The block below finishes the job and installs dbCAN from the maintained AWS S3 release.

Show the complete DRAM repair script (click to expand)

```
cat > "${VP_HOME}/fix_dram.sh" <<'DRAMEOF'
#!/usr/bin/env bash
set -euo pipefail

# Completes the DRAM database after the pipeline's own setup aborts on dbCAN.
VP_HOME="${VP_HOME:-$(pwd)}"
DRAM_DB="${VP_HOME}/db/dram"
RAW="${DRAM_DB}/database_files"
IMG="${VP_HOME}/nextflow-singularity-cache/denglab-viroprofiler-geneannot-v0.2.img"

# Pinned dbCAN release. Listing: https://dbcan.s3.us-west-2.amazonaws.com/
DBCAN_URL="https://dbcan.s3.us-west-2.amazonaws.com/db_v5-2-9_5-5-2026/dbCAN.hmm"

# HOME is exported inside the shell: singularity refuses --env HOME with
# "Overriding HOME environment variable with SINGULARITYENV_HOME is not permitted".
in_container() {
    singularity exec --writable-tmpfs -B "${VP_HOME}" "${IMG}" \
        bash -c "export HOME='${VP_HOME}/container-home'; $1"
}

[[ -f "${IMG}" ]]      || { echo "ERROR: container image missing: ${IMG}" >&2; exit 1; }
[[ -d "${DRAM_DB}" ]]  || { echo "ERROR: no DRAM dir at ${DRAM_DB}" >&2; exit 1; }
cp -a "${DRAM_DB}/CONFIG" "${DRAM_DB}/CONFIG.bak-$(date +%Y%m%d-%H%M%S)"

echo "== removing HTML files masquerading as dbCAN data =="
for f in dbCAN-HMMdb-V11.txt CAZyDB.08062022.fam-activities.txt CAZyDB.08062022.fam.subfam.ec
    if [[ -f "${DRAM_DB}/${f}" ]] && head -c 15 "${DRAM_DB}/${f}" | grep -q "DOCTYPE html"; t
        rm -f "${DRAM_DB}/${f}"; echo "   removed ${f}"
    fi
done

echo "== RefSeq viral proteins =="
if [[ ! -f "${DRAM_DB}/viral.mmsdb.dbtype" ]]; then
    in_container "mmseqs createdb '${RAW}/viral.merged.protein.faa.gz' '${DRAM_DB}/viral.mmsd
fi

echo "== MEROPS peptidases =="
if [[ ! -f "${DRAM_DB}/peptidase.mmsdb.dbtype" ]]; then
    in_container "mmseqs createdb '${RAW}/merops_peptidases_nr.faa' '${DRAM_DB}/peptidase.mms
fi

echo "== VOGdb HMMs =="
if [[ ! -f "${DRAM_DB}/vog_latest_hmms.txt.h3f" ]]; then
    rm -rf "${DRAM_DB}/vogdb_tmp"; mkdir -p "${DRAM_DB}/vogdb_tmp"
    tar -xzf "${RAW}/vog.hmm.tar.gz" -C "${DRAM_DB}/vogdb_tmp"
    # ~49k files nested under hmm/. find -exec, not a glob: a glob misses the
    # nesting and would exceed ARG_MAX at this count.
    n=$(find "${DRAM_DB}/vogdb_tmp" -name '*.hmm' -type f | wc -l)
    [[ ${n} -gt 0 ]] || { echo "ERROR: no .hmm files in vog.hmm.tar.gz" >&2; exit 1; }
    echo "   concatenating ${n} HMM files"
    : > "${DRAM_DB}/vog_latest_hmms.txt"
```

5.5.3 Verify, and do not skip this

The pipeline reports success while producing broken databases. Each `DB_*` process guards its work with `if [ ! -d <dir> ]`, which is not atomic. If a download fails after creating its directory, every retry sees the directory, prints "database already exists" and exits 0. In practice this produced a **52 KB** VirSorter2 "database" and a DRAM install with no description database, while `du -sh` looked healthy because the large files had downloaded.

Run these checks:

```
cd "${VP_HOME}"

# 1. Sizes. Anything far below these is incomplete.
du -sh db/*

# 2. VirSorter2 completion marker
ls db/virsorter2/Done_all_setup && echo "VirSorter2 OK"

# 3. DRAM description tables must all be populated
python3 - <<'PYEOF'
import sqlite3, sys, os
db = os.path.join(os.environ.get("VP_HOME", "."), "db/dram/description_db.sqlite")
con = sqlite3.connect(db)
expect = {"pfam_description": 1000, "viral_description": 10000,
          "peptidase_description": 10000, "vogdb_description": 1000}
bad = False
for t, floor in expect.items():
    n = con.execute(f"SELECT count(*) FROM {t}").fetchone()[0]
    flag = "OK  " if n >= floor else "FAIL"
    if n < floor: bad = True
    print(f"  [{flag}] {t}: {n:,} rows")
n = con.execute("SELECT count(*) FROM dbcan_description").fetchone()[0]
print(f"  [note] dbcan_description: {n:,} rows (0 is expected, see @sec-dram-database-dead-dbca
sys.exit(1 if bad else 0)
PYEOF

# 4. Every database registered in DRAM's CONFIG must exist on disk
python3 - <<'PYEOF'
import json, os
cfg = json.load(open(os.path.join(os.environ.get("VP_HOME", "."), "db/dram/CONFIG")))
for k in ["kofam_hmm", "kofam_ko_list", "pfam", "viral", "peptidase", "vogdb", "dbcan"]:
    v = cfg["search_databases"].get(k)
    print(f"  [{'OK  ' if v and os.path.exists(v) else 'FAIL'}] {k}")
PYEOF
```

Expected sizes:

```
7.0G    db/checkv
```

```
203G    db/dram
199G    db/iphop
4.0G    db/taxonomy
11G     db/vibrant
11G     db/virsorter2
```

and all description tables populated:

```
  [OK  ] pfam_description: 30,134 rows
  [OK  ] viral_description: 722,107 rows
  [OK  ] peptidase_description: 1,227,939 rows
  [OK  ] vogdb_description: 49,116 rows
  [note] dbcan_description: 0 rows (0 is expected, see @sec-dram-database-dead-dbcan-downloads)
```

> **Caution**
>
> `du -sh` alone is **not** a sufficient check. DRAM looked like a healthy 193 GB while missing its description database entirely, because kofam and pfam, the two largest components, had downloaded successfully.

5.6 Quick test run

Before committing real data and days of compute, confirm the installation works end to end. The pipeline ships a small bundled test dataset for exactly this.

5.6.1 Run it

```
cd "${VP_HOME}"
conda activate viroprofiler_env
export NXF_VER=25.10.2

# Update the pipeline
nextflow pull deng-lab/viroprofiler

# Run the bundled test
nextflow run deng-lab/viroprofiler -r main \
    -profile singularity,test \
    --db "${VP_HOME}/db" \
    -c "${VP_HOME}/local.config" \
    -work-dir "${VP_HOME}/work" \
    --outdir "${VP_HOME}/results_test"
```

The `test` profile supplies its own five small samples (HT02, HT04, UC20, UC21, UC24) from a public repository, so you need no input of your own.

5.6.2 How long it takes

Allow 6 to 9 hours. The first run is the slowest because every container image is downloaded and converted to SIF.

Stage	Typical time
Container downloads, first run only	30 to 60 min
FastQC, fastp, assembly of 5 tiny samples	10 to 20 min
CheckV, VIBRANT, VirSorter2, DeepVirFinder	30 to 60 min
DRAM-v	10 to 50 min
vConTACT2	**2 to 5 h**
iPHoP	**1 to 3 h**

vConTACT2 and iPHoP dominate, and neither scales with how small the test input is. vConTACT2 merges your contigs into the entire ProkaryoticViralRefSeq database of about 421,000 protein profiles, so its runtime is driven by the reference. iPHoP searches a 199 GB database. A test dataset of 46 contigs still took about 5 hours for vConTACT2 alone on a single thread.

Monitor it from a second terminal:

```
cd "${VP_HOME}"
tail -f .nextflow.log | grep --line-buffered "Submitted process\|Task completed"
```

5.6.3 What you should see

A successful run ends with:

```
-[ViroProfiler] Pipeline completed successfully-
```

and produces these directories under `results_test/`:

```
ls "${VP_HOME}/results_test"
```

```
abundance   bacphlip   checkv     contigindex  contiglib  dramv
dvf         fastp      fastqc     genepred4ctg mapping2contigs2
multiqc     nrgene     nrprot     pipeline_info results
spades      taxonomy   vibrant    viralhost    vircontigs   virsorter2
```

Quick checks that the biology came through, not just the exit code:

```
cd "${VP_HOME}/results_test"

# vOTUs recovered
zcat contiglib/contigs_cclib_long.fasta.gz | grep -c "^>"

# genome quality distribution
cut -f8 checkv/quality_summary.tsv | sort | uniq -c | sort -rn

# viral contigs flagged
tail -n +2 virsorter2/vs2_category.csv | wc -l

# gene annotations, including CAZymes if the DRAM repair worked
head -1 dramv/dramv-annotate/annotations.tsv | tr '\t' '\n' | grep -n cazy
```

Because the test dataset is deliberately tiny, expect **tens of vOTUs**, mostly CheckV `Low-quality`, and few or no taxonomic or host assignments. That is a correct result. You are testing that the machinery runs, not doing biology.

> **Note**
>
> **If DRAMV fails here**, the DRAM database is incomplete. Return to section 4.2. If VirSorter2 fails, check `db/virsorter2/Done_all_setup` exists.

5.7 A full analysis run

This is the run that produced the results and timings quoted throughout this guide. Use it as the template for real data.

5.7.1 Where the reads come from

Two VLP enriched viromes from ENA study PRJDB10879, downloaded directly over HTTPS from the ENA mirror. No SRA toolkit and no account needed.

Run	Read pairs	Download
DRR270513	2,406,326	about 475 MB
DRR270515	2,969,901	about 569 MB

Why these and not something smaller. Two earlier attempts failed, each after hours of compute, and they illustrate the two ways a viral pipeline dies on input that looks perfectly reasonable:

Attempt	Reads	What happened
ERR14747893	200k pairs	Almost entirely adapter. fastp discarded **100%** as `too_short`, SPAdes died on an empty file
ERR15117916 + ERR15117121	296k + 488k pairs	Reads were **good**, 97 to 98% survived fastp, but far too shallow. Longest contigs 672 bp and 1,668 bp, **zero** reached the 3,000 bp threshold, CheckV then failed on an empty FASTA

Good reads are not sufficient. A viral pipeline needs assembly depth. VLP enriched viromes assemble well at moderate depth because viral genomes are abundant and community complexity is low. DRR270513 was assembly checked before being adopted here:

```
77,316 contigs | longest 41,322 bp | 900 >= 3,000 bp | 97 >= 10 kb
```

5.7.2 Download the reads

```
cd "${VP_HOME}"
mkdir -p data/test

ENA="https://ftp.sra.ebi.ac.uk/vol1/fastq"
for acc in DRR270513 DRR270515; do
  pre="${acc:0:6}"
  for r in 1 2; do
    f="data/test/${acc}_${r}.fastq.gz"
    [ -s "$f" ] && { echo "   $f present"; continue; }
    echo "   fetching ${acc}_${r}.fastq.gz"
    curl -fSL --retry 3 --retry-delay 5 --max-time 3600 -o "${f}.part" \
      "${ENA}/${pre}/${acc}/${acc}_${r}.fastq.gz"
    # Validate content, not the HTTP status: mirrors sometimes serve an HTML
    # error page with a 200 response.
    if [ "$(file -b --mime-type "${f}.part")" != "application/gzip" ]; then
      rm -f "${f}.part"; echo "ERROR: ${acc}_${r} is not gzip data" >&2; exit 1
    fi
    mv "${f}.part" "$f"
  done
done

du -sh data/test
```

5.7.3 Check the reads before committing

This takes two minutes and would have saved both failed attempts above.

```
cd "${VP_HOME}"
FASTP_IMG=$(ls nextflow-singularity-cache/*fastp*.img | head -1)

singularity exec -B "${VP_HOME}" "${FASTP_IMG}" fastp \
  -i data/test/DRR270513_1.fastq.gz -I data/test/DRR270513_2.fastq.gz \
  -o /tmp/o1.fq.gz -O /tmp/o2.fq.gz -j /tmp/qc.json -h /tmp/qc.html --thread 4

python3 - <<'PYEOF'
import json
d = json.load(open("/tmp/qc.json")); s = d["summary"]
b, a = s["before_filtering"]["total_reads"], s["after_filtering"]["total_reads"]
print(f"{b:,} -> {a:,} reads ({100*a/b:.1f}% pass), read length {s['before_filtering']['read1_m
PYEOF
rm -f /tmp/o1.fq.gz /tmp/o2.fq.gz
```

A healthy result keeps most reads. If the pass rate is near zero, the accession is adapter or otherwise unusable, and nothing downstream will work.

5.7.4 Write the samplesheet

The input is a three column CSV. Paths may be local or URLs.

```
cd "${VP_HOME}"
cat > samplesheet.csv <<EOF
sample,fastq_1,fastq_2
DRR270513,${VP_HOME}/data/test/DRR270513_1.fastq.gz,${VP_HOME}/data/test/DRR270513_2.fastq.gz
DRR270515,${VP_HOME}/data/test/DRR270515_1.fastq.gz,${VP_HOME}/data/test/DRR270515_2.fastq.gz
EOF

cat samplesheet.csv
```

Sample names cannot contain spaces, and files must end `.fastq.gz` or `.fq.gz`.

5.7.5 Run the analysis

```
cd "${VP_HOME}"
conda activate viroprofiler_env
export NXF_VER=25.10.2

nextflow run deng-lab/viroprofiler \
```

```
    -r main -profile singularity \
    -c "${VP_HOME}/local.config" \
    -work-dir "${VP_HOME}/work_analysis" \
    --input "${VP_HOME}/samplesheet.csv" \
    --outdir "${VP_HOME}/results_analysis" \
    --db "${VP_HOME}/db" \
    -resume
```

> **Warning**
>
> Use `-profile singularity` **without** `,test` here. The `test` profile would replace your input with the bundled samplesheet and cap resources at 2 CPUs and 6 GB.

To keep it running after you close the terminal:

```
nohup nextflow run deng-lab/viroprofiler ... > run.log 2>&1 &
tail -f run.log
```

5.7.6 How long it takes

The reference run took **10 hours 53 minutes** on 12 cores, 31 GB RAM and 32 GB swap. Measured per stage, from `results_analysis/pipeline_info/execution_trace_*.txt`:

Process	Duration	CPU used	Peak RAM
VIRALHOST_IPHOP	8 h 11 m	399%	**33.7 GB**
TAXONOMY_VCONTACT	1 h 51 m	660%	9.9 GB
DRAMV	51 m	409%	25.4 GB
VIBRANT	43 m	100%	0.4 GB
BACPHLIP	40 m	101%	0.1 GB
SPADES (DRR270515)	29 m	750%	10.4 GB
CHECKV	29 m	100%	2.3 GB
SPADES (DRR270513)	26 m	742%	9.8 GB

Plan for **10 to 14 hours** on comparable hardware, and longer if you have fewer cores or skip the swap step. Both long stages scale with the number of viral contigs found rather than with input file size, so a richer sample takes longer.

5.7.7 What the run produced

```
vOTUs in the non-redundant library (>= 3 kb)   1,795
Contigs flagged viral by VirSorter2            1,608
Genomes assessed by CheckV                     1,779
Contigs with a taxonomic assignment            1,321
Viruses with a predicted host (genus, >= 90%)    200
```

```
Genes annotated by DRAM-v                    15,405
Auxiliary metabolic genes                       357
CAZymes (dbCAN)                                  82
```

CheckV quality distribution:

Tier	Count
Complete	6
High-quality	11
Medium-quality	32
Low-quality	1,550
Not-determined	180

Top CAZyme families, a useful check that the dbCAN repair worked and is biologically sensible:

Family	Count	What it is
GH24	26	Phage lysozyme (endolysin)
GH108	13	Peptidoglycan hydrolase
GT11	8	Fucosyltransferase
GH19	7	Chitinase / lysozyme
GT2	6	Glycosyltransferase
GH23	4	Peptidoglycan lyase

Four of the top six degrade peptidoglycan. These are lysis enzymes, which is exactly what phage genomes should carry.

Section 11 explains how to read each of these outputs, and the thresholds to apply before drawing conclusions from them.

5.8 Known issues and how this setup fixes them

Each of these is a real failure that was diagnosed and fixed. They are recorded because you will hit them if you deviate from these scripts.

5.8.1 VirSorter2: read-only `$HOME` in the container

```
[Errno 30] Read-only file system: '/home/<user>/.virsorter'
```

Nextflow launches Singularity with `--no-home`, so `$HOME` inside the container is the image's read-only layer. VirSorter2 insists on creating `~/.virsorter` and `~/.cache/conda`.

Fix: redirect `HOME` into the project (`local.config`):

```
env { HOME = "${VP_HOME}/container-home" }
```

Do not instead bind-mount over `/home/<user>`. Nextflow separately binds the pipeline's own scripts from `~/.nextflow/assets/deng-lab/viroprofiler/bin` and puts them on `PATH`; mounting anything onto your home directory hides that bind and every process calling a pipeline script dies with `run_checkv.sh: command not found` (exit 127).

5.8.2 DRAM-v: read-only container filesystem

```
ln: failed to create symbolic link '/opt/conda/db2': Read-only file system
```

DRAM hardcodes its database path to `/opt/conda/db2`, so the pipeline symlinks your database there at runtime. That works under Docker (writable upper layer) but not under Singularity, where the image is read-only squashfs.

Fix: give the container a small in-memory overlay:

```
singularity { runOptions = "-B ${VP_HOME} --writable-tmpfs" }
```

Binding over `/opt/conda/db2` does *not* work: the script calls `ln -s` unconditionally, so a pre-existing path fails with `File exists`.

5.8.3 DRAM database: dead dbCAN downloads

DRAM 1.4.6 downloads three dbCAN/CAZy files from `bcb.unl.edu`. That host now answers **every** path under `/dbCAN2/download/` with the dbCAN3 web app - **HTTP 200, `text/html`, 19313 bytes**. DRAM saves the HTML, `hmmpress` fails:

```
Error: File format problem ... dbCAN-HMMdb-V11.txt
Format tag is '<!DOCTYPE': unrecognized.
```

Setup then aborts *before* building the description database, and DRAM-v later crashes with `AttributeError: 'NoneType' object has no attribute 'query'`.

Fix: `setup_databases.sh` fetches dbCAN from the maintained AWS S3 release instead, and validates content rather than HTTP status.

> **Note**
>
> **Known limitation.** The current dbCAN distribution no longer ships the legacy `CAZyDB.*.fam-activities.txt` descriptions file that DRAM 1.4.6 expects. CAZy **family**

IDs are annotated (e.g. `GH24`, phage lysozyme), but the long description and subfamily-EC columns are blank. Nothing is fabricated to fill this gap. `dbcan_description: 0 rows` is expected.

5.8.4 iPHoP: out-of-memory, and the wrong fix for it

iPHoP's stage 6 (RaFAH) runs a random forest in R that replicates its data per thread, so raising `cpus` raises peak memory. Every row here is measured:

Threads	Swap	Peak RSS	Outcome
1	none	19.3 GB	killed
4	none	20.5 GB	killed
8	32 GB	**33.7 GB**	**completes in 8 h 11 m**

The instructive part is the fix that did not work. The obvious response to "more threads uses more memory" is to pin it to one CPU. That was done, and it was wrong: only stage 6 of 6 is memory-hungry, while stages 1-5 (blastn, CRISPR blastn, WIsH, VHM, PHP) are CPU-bound and thread well. Throttling all six stages to protect one left blastn **4 hours into stage 1 with 11 cores idle**.

What actually works is moderate threading plus swap:

- `cpus = 8`: fast through stages 1-5
- 32 GB swap: absorbs stage 6's peak, which **exceeds physical RAM** (33.7 GB on a 31 GB machine)
- a retry ladder, because 8 threads was an extrapolation from the 1- and 4-thread measurements, not itself a measurement:

```
cpus = { task.attempt == 1 ? 8 : (task.attempt == 2 ? 4 : 1) }
errorStrategy = { task.exitStatus in [104,134,137,139,143,247,251] && task.attempt <= 3 ? 'retr
maxRetries = 2
```

`137` is `128+9`: SIGKILL, which is what the OOM killer sends. If 8 threads is too much on your machine, the task steps down to 4 and then to 1 automatically rather than failing the run. In the reference run the ladder never fired.

Retrying is safe *here* because iPHoP writes into a fresh task directory on each attempt: unlike the `DB_*` processes (Section 5.5.3), where a retry short-circuits on a non-atomic `if [ ! -d ]` guard and hides the failure.

> **!** Important
>
> A Nextflow `memory` directive governs Nextflow's **scheduling** only. It cannot reserve RAM from the OS, from your browser, or invent 33.7 GB on a 31 GB machine. Only swap does that.

5.8.5 Every process silently gets 1 CPU and 10 GB

This is the single most consequential issue in the whole setup, and it caused four separate failures before being understood.

ViroProfiler's processes carry only *container* labels (`viroprofiler_base`, `viroprofiler_host`, …). The nf-core resource labels (`process_low/medium/high`) are either absent or do not take effect, so processes fall back to the pipeline's default of **1 CPU / 10 GB**. Observed consequences:

Process	What it actually received	Result
SPADES	`--threads 1 --memory 10` *despite* `label 'process_high'`	died at its **own** 10 GB ceiling: `mmap(2) failed. Reason: Cannot allocate memory`
VIRALHOST_IPHOP	`--num_threads 1`	4 hours on stage 1 of 6
TAXONOMY_VCONTACT	`-t 1`	~5 hours
DRAMV	1 thread	minutes of work stretched out

The SPAdes case is worth dwelling on: the error *looks* like the machine ran out of RAM, but 19 GB was free at the time. SPAdes passes the Nextflow `memory` directive straight through as its own `--memory` ceiling, so an under-declaration becomes a hard cap that aborts the assembly.

`params.max_cpus` / `max_memory` cannot fix any of this. `check_max()` is a ceiling applied to a request: it only caps values *downward*, never raises them. Only an explicit `withName:` block does, and `withName` has the highest precedence of any Nextflow selector.

`local.config` therefore assigns every process explicitly, in tiers scaled to the detected machine (see the appendix for the full file):

```
withName: 'SPADES'             { cpus = CPU_HEAVY;  memory = MEM_HEAVY; maxForks = 1 }
withName: 'TAXONOMY_VCONTACT' { cpus = CPU_HEAVY;  memory = MEM_HEAVY; maxForks = 1 }
withName: 'DRAMV'              { cpus = CPU_HEAVY;  memory = MEM_HEAVY }
withName: 'VIRALHOST_IPHOP'    { cpus = CPU_IPHOP;  memory = MEM_HEAVY; maxForks = 1 }
withName: 'CHECKV|VIBRANT|VIRSORTER2|...' { cpus = CPU_MEDIUM; memory = MEM_MEDIUM }
withName: 'FASTQC|FASTP|BACPHLIP|...'     { cpus = CPU_LIGHT;  memory = MEM_LIGHT  }
```

Measured effect of doing this:

Process	Default (1 CPU)	Explicit allocation
vConTACT2	~5 h	**1 h 51 m** (660% CPU)
iPHoP	4 h on stage 1 alone	all 6 stages in 8 h 11 m (399% CPU)
SPAdes	aborted at 10 GB	26-29 min (750% CPU)
DRAM-v	single-threaded	51 m (409% CPU)

Verify it took effect in the first minutes of a run: this is the check that would have caught all four failures immediately:

```
grep -oE '(--threads|--num_threads|-t) [0-9]+' work/*/*/.command.sh | tail
```

If you see `1` where you expect more, the selector did not match.

> **Tip**
>
> **Do not give everything the whole machine.** VIBRANT, BACPHLIP and CheckV run at ~100% CPU in the trace: they are single-threaded internally, so extra cores are wasted on them and merely block other tasks from being scheduled. That is why the medium tier is half the machine, not all of it.

5.8.6 vConTACT2 is slow, and that is normal

vConTACT2 merges your contigs into the **entire** ProkaryoticViralRefSeq database (~421,000 protein profiles), so its runtime is driven by the reference, not by your input size. A 46-contig test dataset still took ~**5 hours** when running single-threaded.

With the explicit allocation it took **1 h 51 m at 660% CPU** on 1,795 vOTUs - i.e. **far more data in a third of the time**. Threads help the Diamond all-vs-all stage; profile-building and ClusterONE remain single-threaded, which is why it never scales linearly and stays an hours-long stage regardless.

Do not interpret a long vConTACT2 runtime as a hang. Check progress with:

```
tail -3 work/<hash>/.command.err
```

It prints stage markers such as `Building the cluster and profiles`, and can sit in one stage for a long time while writing nothing.

5.8.7 `-resume` and session IDs

Bare `-resume` resumes the **most recent** session. Any other Nextflow invocation in the same directory: even a no-op `-preview`: becomes "most recent" and silently redirects your resume to a session with no cached work, re-running everything.

```
nextflow log                                # list sessions with IDs
nextflow run ... -resume <session-uuid>     # pin explicitly
```

Also note the `DB_*` setup processes declare **no outputs**, so deleting a database directory does not invalidate their cache. Never use `-resume` to repair a database: run without it.

5.9 Running your own data

5.9.1 Samplesheet

A CSV with three columns. Paths may be absolute local paths or URLs.

```
sample,fastq_1,fastq_2
SampleA,/data/SampleA_R1.fastq.gz,/data/SampleA_R2.fastq.gz
SampleB,/data/SampleB_R1.fastq.gz,/data/SampleB_R2.fastq.gz
```

Sample names cannot contain spaces; files must end `.fastq.gz` or `.fq.gz`.

5.9.2 Run

```
conda activate nextflow_viroprofiler_env
export VP_HOME=/path/to/viroprofiler

NXF_VER=25.10.2 nextflow run deng-lab/viroprofiler \
    -r main -profile singularity \
    -c "${VP_HOME}/local.config" \
    --input samplesheet.csv \
    --outdir results \
    --db "${VP_HOME}/db" \
    --max_cpus 10 --max_memory 24.GB --max_time 48.h \
    -resume
```

5.9.3 Check your reads first

Run fastp on new accessions before committing to a pipeline run. A plausible-looking public accession (ERR14747893, 200,040 reads) turned out to be almost entirely adapter: fastp discarded **100%** of it as `too_short`, and SPAdes died hours later on an empty input file:

```
== Error ==  file is empty: ERR14747893_1.fastp.fastq.gz
```

```
singularity exec nextflow-singularity-cache/depot.galaxyproject.org-singularity-fastp-0.23.2--h
  fastp -i reads_1.fastq.gz -I reads_2.fastq.gz -o /tmp/o1.gz -O /tmp/o2.gz -j qc.json -h qc.ht
```

Then check `summary.after_filtering.total_reads` is a sensible fraction of `before_filtering`.

5.9.4 Useful options

Option	Meaning
`--mode setup`	Download databases only
`--mode all`	Full analysis (default)
`--use_dram false`	Skip DRAM-v (saves time and 203 GB)
`--use_iphop false`	Skip host prediction (saves 199 GB)
`--assemblies scaffolds`	Use scaffolds rather than contigs
`--contig_minlen 3000`	Minimum contig length
`-work-dir DIR`	Where intermediate files go
`-resume <id>`	Reuse cached results from a session

5.10 Monitoring a running pipeline

Runs take hours to days, so you will want to check on them from another terminal. Everything below only reads files and can never disturb the run.

5.10.1 Is it alive, and what is it doing?

```
cd "${VP_HOME}"

# Live feed of tasks starting and finishing
tail -f .nextflow.log | grep --line-buffered "Submitted process\|Task completed"

# Currently submitted tasks
grep "Submitted process >" .nextflow.log | tail -5

# Is the Nextflow process still running at all?
ps -eo etime,cmd | grep "[n]extflow.*one.jar"
```

5.10.2 Confirm each tool got the CPUs you assigned

This is the fastest way to catch the Section 5.8.5 problem, where a process silently falls back to 1 CPU. Run it in the first few minutes.

```
cd "${VP_HOME}"
grep -ohE '(--threads|--num_threads|-t|--thread) [0-9]+' work*/*/*/.command.sh | sort | uniq -c
```

If you see `1` where you expect more, the `withName` selector did not match that process.

5.10.3 Which internal stage a tool has reached

Nextflow reports a task as "running" for hours without further detail. The tool's own stderr is more informative.

```
cd "${VP_HOME}"

# Find the work directory of the running task, then follow its stderr
grep "Submitted process >" .nextflow.log | tail -1
tail -5 work*/<hash>/.command.err
```

iPHoP prints stage markers such as `[1/1/Run] Running blastn against genomes...` through `[6/2/Run]`, so you can see which of its six stages it is on. vConTACT2 prints markers such as `Building the cluster and profiles`.

5.10.4 System resources

```
free -h                 # RAM and swap in use
swapon --show           # confirm swap is active
uptime                  # load average, compare with your core count
df -h "${VP_HOME}"      # disk headroom
dmesg -T | grep -i "killed process"   # confirm an OOM kill vs a tool error
```

5.10.5 After the run

```
firefox "${VP_HOME}"/results_analysis/pipeline_info/execution_report_*.html
```

The execution report gives per-process runtime, CPU percentage and peak RAM, which is the best guide to what to tune next time.

> **Note**
>
> **Low CPU is not necessarily a stall.** Several stages are internally single-threaded: iPHoP's final aggregation loop, vConTACT2's ClusterONE step, VIBRANT and BACPHLIP. Seeing one busy core while the tool was given 8 threads is normal, not a contradiction.

5.11 Output structure

This is the actual layout produced by a completed run (`--outdir`):

```
output/
   fastqc/          per-sample read quality reports
   fastp/           trimming reports and filtered reads
   spades/          per-sample assemblies
   contiglib/       non-redundant viral contig library (vOTUs)
   contigindex/     Bowtie2 index of the library
   mapping2contigs2/ per-sample BAMs
   checkv/          genome quality and completeness
   virsorter2/      viral identification
   vibrant/         viral identification + annotation
   dvf/             DeepVirFinder scores
   vircontigs/      putative viral contigs
   genepred4ctg/    predicted genes
   nrprot/ nrgene/  non-redundant protein / gene sets
   abundance/       per-sample abundance tables
   taxonomy/        vConTACT2 and MMseqs2 assignments
   dramv/           gene annotation, AMG summaries
   viralhost/       iPHoP host predictions
   bacphlip/        lifestyle predictions
   results/         TreeSummarizedExperiment objects (.rds)
   multiqc/         aggregated QC report
   pipeline_info/   execution report, timeline, DAG
```

Key files:

File	Contents
`contiglib/contigs_cclib_long.fasta.gz`	The vOTU sequences (length-filtered)
`contiglib/contigs_ANIclst.tsv`	Clustering assignments (which contigs form each vOTU)
`checkv/quality_summary.tsv`	Completeness, contamination, quality tier
`checkv/checkv_qc_long.fasta`	Quality-passing viral genomes
`abundance/abundance_contigs_tpm.tsv.gz`	vOTU × sample abundance (TPM)
`abundance/abundance_contigs_count.tsv.gz`	Raw read counts
`abundance/abundance_contigs_covered_fraction_tsv.gz`	Breadth of coverage: use this to filter spurious hits
`taxonomy/taxonomy.tsv`	Merged taxonomic assignments
`taxonomy/out_vContact2/`	vConTACT2 gene-sharing network output
`dramv/dramv-distill/amg_summary.tsv`	Auxiliary metabolic genes
`dramv/dramv-annotate/annotations.tsv`	Per-gene annotations (incl. `cazy_ids`)
`viralhost/out_iphop/Host_prediction_to_genus_m90.csv`	Predicted hosts (genus, 90% confidence)
`virsorter2/vs2_category.csv`	VirSorter2 categories
`results/viroprofiler_output.rds`	TreeSummarizedExperiment for R

File	Contents
`multiqc/multiqc_report.html`	Read QC overview
`pipeline_info/execution_report_*.html`	Runtime, CPU and memory per process

The `results/*.rds` files are TreeSummarizedExperiment objects: load with `readRDS()` in R for downstream analysis.

> **Tip**
>
> **Interpretation tip.** Always filter on `abundance_contigs_covered_fraction` before trusting an abundance value. A high TPM with low breadth of coverage usually means reads piled onto one conserved region, not a genuinely present genome.

5.12 Interpreting your results

This section explains what each output actually means, in the order you should read them. All numbers quoted are from the reference run.

5.12.1 Start here: a five-minute triage

```
cd results_test

# 1. How many vOTUs did we recover?
zcat contiglib/contigs_cclib_long.fasta.gz | grep -c "^>"

# 2. How good are they?
cut -f8 checkv/quality_summary.tsv | sort | uniq -c | sort -rn

# 3. Did anything get a name?
awk -F'\t' 'NR>1 && $6!=""' taxonomy/taxonomy.tsv | wc -l

# 4. Read QC sanity
firefox multiqc/multiqc_report.html
```

If step 1 returns a small number, or step 2 is overwhelmingly `Not-determined`, stop and look at your assembly before interpreting anything downstream.

5.12.2 The vOTU library: your unit of analysis

`contiglib/contigs_cclib_long.fasta.gz` holds the **non-redundant viral contig library**. Contigs from all samples are pooled and clustered, so each sequence is one **vOTU**: operationally a viral "species": and every sample is then quantified against this shared set. That is what makes cross-sample comparison valid.

`contiglib/contigs_ANIclst.tsv` records which original contigs collapsed into each vOTU.

> **Tip**
>
> **Contig names encode their origin**, e.g. `DRR270513__NODE_1_length_54666_cov_12.277197`: the sample it assembled in, the assembler node, its **length**, and its **assembly coverage**. Length and coverage are useful sanity checks: a 54 kb contig at 12× is a credible phage genome; a 3 kb contig at 1.5× is not.

5.12.3 CheckV: how much of a genome do you actually have?

`checkv/quality_summary.tsv` is the most important quality file in the run. Reference run distribution:

Tier	Count	Meaning
Complete	6	Genome ends detected (DTR/ITR); a finished genome
High-quality	11	90% estimated completeness
Medium-quality	32	50-90% complete
Low-quality	1,550	<50% complete: fragments
Not-determined	180	Too little information to estimate

A long low-quality tail is normal and is not a failure. Most assembled viral contigs are genome fragments. What matters is matching the analysis to the quality tier:

Question	Minimum tier
Abundance / ecology across samples	Any tier (all vOTUs)
Gene content, AMGs, functional claims	Medium+
Genome architecture, lifestyle, publication-grade genomes	High-quality or Complete

Useful columns: `completeness`, `contamination`, `provirus` (integrated prophage), `warnings`, and `miuvig_quality`: the latter maps to the community MIUViG reporting standard, which is what reviewers will expect you to cite.

```
# the genomes worth treating as genomes
awk -F'\t' 'NR==1 || $8=="Complete" || $8=="High-quality"' \
    checkv/quality_summary.tsv > high_conf_vOTUs.tsv
```

5.12.4 Abundance: who is there, and how much?

`abundance/` holds one matrix per metric, all vOTUs × samples:

File	Use it for
`abundance_contigs_tpm.tsv.gz`	Comparing across samples (length- and depth-normalised)
`abundance_contigs_count.tsv.gz`	Raw counts: input to DESeq2/edgeR, which want counts
`abundance_contigs_covered_fraction.tsv.gz`	**Presence/absence filtering**
`abundance_contigs_rpkm.tsv.gz`	Legacy normalisation
`abundance_contigs_trimmed_mean.tsv.gz`	Coverage robust to uneven pileups

> **Caution**
>
> **Always filter on breadth of coverage before trusting abundance.** A high TPM with low `covered_fraction` usually means reads piled onto one conserved region: a phage tail fibre, a transposase: not that the genome is present. A common threshold is **70-75% of the contig covered**; below that, treat the vOTU as absent in that sample.

```
import pandas as pd
tpm   = pd.read_csv("abundance/abundance_contigs_tpm.tsv.gz", sep="\t", index_col=0)
brdth = pd.read_csv("abundance/abundance_contigs_covered_fraction.tsv.gz", sep="\t", index_col=
tpm_filtered = tpm.where(brdth >= 0.75, 0.0)      # zero out low-breadth calls
```

5.12.5 Taxonomy: expect most of it to be unclassified

`taxonomy/taxonomy.tsv` merges two independent approaches, and they answer different questions:

- **MMseqs2** columns (`Kingdom` … `Species`): homology to known viral proteins.
- **vConTACT2** columns (`VC`, `vc_*`): gene-sharing network clustering. A shared **VC** is roughly a **genus-level** grouping.

Reference run, of 1,795 vOTUs:

Level	Assigned
Kingdom	1,288
Family	993
vConTACT2 VC	**145**

This is a normal result, not a failure. Most environmental viruses have no close cultured relative: the "viral dark matter" problem. Empty taxonomy rows mean *no confident match*, not a broken pipeline.

Practical reading: use MMseqs2 ranks for a broad picture (`Caudoviricetes` will dominate most bacteriophage datasets), and treat a shared `VC` as the stronger evidence that two vOTUs are genuinely related. vOTUs in a VC **with** reference genomes inherit a credible genus assignment; VCs of only your own contigs are candidate novel genera.

5.12.6 Host prediction: which bacteria do these phages infect?

`viralhost/out_iphop/Host_prediction_to_genus_m90.csv`. In the reference run, **200 of 1,795 vOTUs** got a host: a typical yield.

```
Virus, AAI to closest RaFAH reference, Host genus, Confidence score, List of methods
DRR270513__NODE_102..., 22.89, d__Bacteria;...;g__BACL21, 93.1
```

- **Host genus** uses GTDB taxonomy, so it is directly comparable with a 16S/metagenomic profile of the same samples.
- **Confidence score**: the `m90` file is already filtered to 90, meaning an estimated 10% false-discovery rate at genus level. Do not lower it casually.
- **List of methods**: agreement of several independent methods (CRISPR, blast, WIsH, PHP, RaFAH) is stronger evidence than a single one.

The natural next step is to check whether predicted hosts are actually present in your samples. Consistency between a phage's abundance and its host's is a much stronger claim than either alone.

5.12.7 Lifestyle: read this one with care

`bacphlip/*.bacphlip` gives per-contig `Virulent` and `Temperate` probabilities.

> **Caution**
>
> **Important caveat found in this run:** 1,010 of 1,751 contigs share the *identical* value `0.8118782`: that is BACPHLIP's prior when the genome contains **no informative HMM hits**, which is what happens on short, incomplete contigs. Those rows carry no information. BACPHLIP is designed for **complete genomes**. Restrict lifestyle claims to CheckV Complete/High-quality vOTUs, and discard the repeated default value.

```
# distinct values only; repeated defaults are uninformative
awk 'NR>1{print $2}' bacphlip/*.bacphlip | sort | uniq -c | sort -rn | head
```

A complementary signal: CheckV's `provirus` column flags contigs with detected integration boundaries, which is direct evidence of a temperate lifestyle.

5.12.8 Function, AMGs and CAZymes

`dramv/dramv-annotate/annotations.tsv`: one row per predicted gene. Reference run gave **15,405 genes**.

Column	Source	Reference run
`viral_id`	RefSeq viral	5,084
`pfam_hits`	Pfam	4,053
`peptidase_id`	MEROPS	158
`cazy_ids`	dbCAN	82
`kegg_id`	KOfam	:

Auxiliary metabolic genes: `dramv/dramv-distill/amg_summary.tsv`, 357 in the reference run. AMGs are host-metabolism genes carried by phages, and they are the most over-interpreted output in viral metagenomics.

`auxiliary_score` is the confidence, **lower is better**:

Score	Count	Interpretation
1	16	Strongest: flanked by confirmed viral genes on both sides
2	142	Strong
3	199	Plausible
4-5	0	Weak: usually discard

> Caution
>
> **AMG calls require manual curation.** A gene on a contig that is really a misclassified bacterial fragment will look like a spectacular AMG. Before making any claim: confirm the contig is confidently viral (CheckV tier, VirSorter2 category), require `auxiliary_score 3`, and inspect `amg_flags` : the `F` flag marks genes near a contig end, where flanking evidence is weak. See the DRAM documentation for the full flag vocabulary.

CAZymes: the families recovered are a good illustration of reading annotations biologically rather than as a list:

Family	Count	What it is
GH24	26	Phage lysozyme (endolysin)
GH108	13	Peptidoglycan hydrolase
GT11	8	Fucosyltransferase
GH19	7	Chitinase / lysozyme
GT2	6	Glycosyltransferase
GH23	4	Peptidoglycan lyase

Four of the top six degrade peptidoglycan: these are **lysis enzymes**, the phage's tool for bursting its host, not host metabolic genes. Glycosyl- transferases (GT11, GT2) more plausibly modify the

phage's own structures or host surface receptors. Reporting "82 CAZymes indicating carbohydrate metabolism" would be wrong; the correct reading is that the CAZyme signal is dominated by lysis machinery.

> **Note**
>
> The `cazy_hits` and `cazy_subfam_ec` description columns are **blank by design**: dbCAN no longer distributes the legacy description file DRAM 1.4.6 expects. Family IDs are complete; only the long text is missing.

5.12.9 Putting it together

Typical questions and the files that answer them:

Question	Files
Which vOTUs differ between groups?	`abundance_contigs_count` → DESeq2, filtered by breadth
Which are novel?	`taxonomy.tsv`: no family, or a VC with no reference members
Which are complete genomes?	`checkv/quality_summary.tsv` tier
Who do they infect?	`Host_prediction_to_genus_m90.csv`
Temperate or lytic?	CheckV `provirus` + BACPHLIP on high-quality genomes only
Do they carry metabolic genes?	`amg_summary.tsv`, score 3, manually curated

For downstream work in R, `results/viroprofiler_output.rds` is a TreeSummarizedExperiment with abundance, taxonomy and per-contig metadata already joined:

```
library(TreeSummarizedExperiment)
tse <- readRDS("results/viroprofiler_output.rds")
assayNames(tse); dim(tse); head(rowData(tse))
```

5.12.10 Interpretation pitfalls

Pitfall	Why it matters
Treating every contig as a genome	86% were <50% complete in the reference run
Trusting TPM without breadth	One conserved region can fake high abundance
Reporting AMGs without curation	Misclassified bacterial contigs produce spurious AMGs
Reading unclassified as failure	Most environmental viruses genuinely have no relatives
Lifestyle calls on fragments	BACPHLIP returns an uninformative default for these

Pitfall	Why it matters
Comparing runs with different databases	Taxonomy and hosts depend on database version: record it

Your database versions and tool versions are recorded in `pipeline_info/`: cite those, not the tool's latest release.

5.13 Troubleshooting

Symptom	Cause	Fix
`Unexpected input: '('` in nextflow.config	Nextflow 26.x	`export NXF_VER=25.10.2`
`run_checkv.sh: command not found`	A bind hides the pipeline's `bin/`	Never mount over `$HOME`; use `env.HOME`
`Read-only file system: '/home/…/.virsorter'`	`--no-home`	Set `env.HOME` to a project path
`ln: … '/opt/conda/db2': Read-only file system`	Read-only squashfs	Add `--writable-tmpfs`
`'NoneType' object has no attribute 'query'`	DRAM `description_db` is null	`bash setup_databases.sh --dram-only`
`KeyError: 'name'` from DRAM	`setup_info` entry lacks `name`	Rebuild CONFIG via `--dram-only`
iPHoP dies, no error in log	OOM-killed	Add swap; keep `cpus = 1`; check `dmesg -T | grep -i "killed process"`
SPAdes: `file is empty: *.fastp.fastq.gz`	fastp removed all reads	Check input quality (Section 5.7.3)
Everything re-runs despite `-resume`	Resumed the wrong session	`-resume <session-uuid>`
`Failed to create Singularity cache directory`	Stale `NXF_SINGULARITY_CACHEDIR`	`source ~/.bashrc` or open a new terminal
"completed successfully" but a database is tiny	Non-atomic `if [ ! -d ]` guard	`bash setup_databases.sh --verify`

5.13.1 Useful commands

```
bash setup_databases.sh --verify                    # check database integrity
nextflow log                                        # list runs and session IDs
```

```
dmesg -T | grep -i "killed process"          # confirm an OOM kill
du -sh db/*                                  # rough size check (not sufficient alone)
cat work/<hash>/.command.err                 # a failed task's stderr
free -h && swapon --show                     # memory and swap
```

5.14 Author and contact

Dr. Muhammad Aammar Tufail

LinkedIn	https://www.linkedin.com/in/aammar-tufail/
GitHub	https://github.com/AammarTufail
Course	Bioinformatics ka Chilla
Email	aammar@codanics.com

This guide, the accompanying scripts, and the database repair procedure were developed and verified on a working ViroProfiler installation. If you use them in your work, a mention is appreciated: and please cite the ViroProfiler authors and the individual tools as set out in the next section.

For questions about the setup, the database repair, or interpreting your results, email aammar@codanics.com or reach out on LinkedIn.

If you are new to bioinformatics and want structured training covering the foundations this pipeline builds on: Linux, Nextflow, sequence analysis and downstream statistics: see Bioinformatics ka Chilla.

5.15 Citation

If you use ViroProfiler, cite the pipeline and the tools it invoked:

> Ru, J., Zhu, Y., Deng, L., et al. "ViroProfiler: a containerized bioinformatics pipeline for viral metagenomic data analysis." *Gut Microbes* 15.1 (2023): 2192522.

Per-tool citations are listed in the pipeline's CITATIONS.md. `results/pipeline_info/` records the exact versions used in your run: cite those.

5.16 Further reading

- ViroProfiler repository
- ViroProfiler wiki
- Nextflow documentation
- Singularity/Apptainer docs
- nf-core

5.17 Appendix: full script listings

Everything below is generated directly from the working scripts, so it always matches the files on disk. With this appendix the guide is self-contained: you can recreate the entire setup from it.

5.17.1 `local.config`

Nextflow configuration. Every fix described in the Known issues section lives here.

Show local.config (231 lines)

```
/*
 * ============================================================================
 * ViroProfiler - local execution config
 * ============================================================================
 * Portable overrides for deng-lab/viroprofiler under Singularity.
 *
 *   nextflow run deng-lab/viroprofiler -c local.config ...
 *
 * PATHS
 *   Nothing is hardcoded. Everything hangs off VP_HOME, which is $VP_HOME if
 *   set, otherwise the directory you launch nextflow from.
 *
 * RESOURCES
 *   CPUs and RAM are DETECTED from the machine and then divided into tiers
 *   below. Override with VP_MAX_CPUS / VP_MAX_MEM_GB if you want to leave more
 *   headroom for other work:
 *       VP_MAX_CPUS=8 VP_MAX_MEM_GB=20 nextflow run ...
 * ============================================================================
 */

def VP_HOME = System.getenv('VP_HOME') ?: System.getProperty('user.dir')

// ----------------------------------------------------------------------------
// Detect the machine, then reserve a little for the OS and desktop.
// ----------------------------------------------------------------------------
def detectedCpus = Runtime.runtime.availableProcessors()

def detectedMemGb = {
    try {
        def line = new File('/proc/meminfo').readLines().find { it.startsWith('MemTotal') }
        return (long) ((line.replaceAll(/\D/, '') as long) / 1024 / 1024)
    } catch (Exception e) {
        return 16L                          // conservative fallback
    }
}()

// Leave 1 core and ~3 GB for the OS: a workstation that is 100% committed to
// the pipeline becomes unusable, and an OOM at 100% commitment kills tasks.
// Swap is the safety net for the peaks, not a substitute for this headroom.
def MAX_CPUS   = (System.getenv('VP_MAX_CPUS')   ?: "${Math.max(1, detectedCpus - 1)}") as in
def MAX_MEM_GB = (System.getenv('VP_MAX_MEM_GB') ?: "${Math.max(8, detectedMemGb - 3)}") as i

// Tiers. Heavy tasks take the machine; light ones run several at a time.
def CPU_HEAVY  = MAX_CPUS
def CPU_MEDIUM = Math.max(2, (int) (MAX_CPUS / 2))
def CPU_LIGHT  = Math.max(2, (int) (MAX_CPUS / 6))   // >=2: fastp/FastQC still thread

def MEM_HEAVY  = "${MAX_MEM_GB}.GB"
def MEM_MEDIUM = "${Math.max(6, (int) (MAX_MEM_GB / 2))}.GB"
def MEM_LIGHT  = "${Math.max(4, (int) (MAX_MEM_GB / 6))}.GB"
```

5.17.2 `install.sh`

Creates the conda environment, installs Nextflow (pinned), and sets up the directory layout.

Show install.sh (133 lines)

```
#!/usr/bin/env bash
set -euo pipefail

# ================================================================================
# ViroProfiler - environment installer
# ================================================================================
#   bash install.sh
#
# Installs the conda environment and Nextflow, prepares the directory layout,
# and persists the Singularity cache settings. It does NOT download databases -
# run setup_databases.sh for that (it needs ~450 GB and several hours).
#
# Everything lives under VP_HOME (default: the current directory), so nothing
# is written to other disks:
#     VP_HOME=/data/viroprofiler bash install.sh
# ================================================================================

VP_HOME="${VP_HOME:-$(pwd)}"
ENV_NAME="${VP_ENV_NAME:-nextflow_viroprofiler_env}"

# Nextflow 26.x CANNOT parse ViroProfiler v0.2.4's nextflow.config: its strict
# config parser rejects the legacy 'def check_max(obj, type) {...}' function
# with "Unexpected input: '('", so the run dies before any process starts.
# Pin to a release that parses it.
NXF_PIN="${NXF_VER:-25.10.2}"

CACHE_DIR="${VP_HOME}/nextflow-singularity-cache"
SING_CACHE_DIR="${VP_HOME}/singularity-cache"
SING_TMP_DIR="${VP_HOME}/singularity-tmp"

echo "Installing ViroProfiler environment under: ${VP_HOME}"

# --- 1. Singularity ---------------------------------------------------------
if ! command -v singularity >/dev/null 2>&1; then
    cat >&2 <<'EOF'
ERROR: singularity not found. Install it, then re-run this script.

  Debian / Ubuntu / Linux Mint:
      sudo apt update && sudo apt install -y singularity-container
  Fedora / RHEL:
      sudo dnf install -y singularity-ce
  Or via conda:
      conda install -c conda-forge singularity

Verify with:  singularity --version
EOF
    exit 1
fi
echo "  singularity: $(singularity --version)"

# --- 2. Conda environment with Nextflow -------------------------------------
```

5.17.3 `setup_databases.sh`

Downloads, repairs and VERIFIES all databases. The verification stage is the part you must not skip.

Show setup_databases.sh (360 lines)

```
#!/usr/bin/env bash
set -euo pipefail

# ================================================================================
# ViroProfiler - database setup and repair
# ================================================================================
# Builds a COMPLETE and VERIFIED ViroProfiler database set on a Linux machine.
#
#   bash setup_databases.sh            # full setup (hours; ~420 GB)
#   bash setup_databases.sh --verify   # check an existing install, change nothing
#   bash setup_databases.sh --dram-only  # skip stage 1, repair DRAM only
#
# WHY THIS SCRIPT EXISTS
# ----------------------
# Running the pipeline's own 'nextflow run ... --mode setup' is NOT sufficient,
# and worse, it reports success when it has silently failed. Two real failures:
#
#   1. VirSorter2 cannot write $HOME inside the container and dies. The
#      pipeline's guard is 'if [ ! -d <dir> ]', which is not atomic, so the
#      failed attempt leaves the directory behind and every retry then prints
#      "database already exists" and exits 0. Result: a 52 KB "database".
#
#   2. DRAM's setup downloads three dbCAN/CAZy files from bcb.unl.edu, which
#      now answers EVERY path under /dbCAN2/download/ with the dbCAN3 web app -
#      HTTP 200, text/html, 19313 bytes. DRAM saves the HTML, hmmpress fails
#      ("Format tag is '<!DOCTYPE': unrecognized"), and setup aborts BEFORE
#      building the description database or processing viral/peptidase/vogdb.
#      DRAM-v then crashes with
#          AttributeError: 'NoneType' object has no attribute 'query'
#      Meanwhile 'du -sh' looks healthy, because kofam and pfam (the huge ones)
#      did download.
#
# So this script validates CONTENT rather than trusting HTTP status codes, and
# verifies the END STATE rather than trusting "completed successfully".
#
# The dbCAN files are fetched from the maintained AWS S3 release instead of the
# dead host. The current dbCAN distribution no longer ships the legacy
# 'fam-activities' descriptions file that DRAM 1.4.6 expects; CAZy family IDs
# are therefore annotated, but their long descriptions are left blank. Nothing
# is fabricated to fill that gap.
# ================================================================================

VP_HOME="${VP_HOME:-$(pwd)}"
DB_DIR="${VP_HOME}/db"
DRAM_DB="${DB_DIR}/dram"
RAW="${DRAM_DB}/database_files"
CACHE_DIR="${VP_HOME}/nextflow-singularity-cache"
WORK_DIR="${VP_HOME}/work"
MAX_CPUS="${VP_MAX_CPUS:-10}"
MAX_MEM="${VP_MAX_MEM:-24.GB}"
NXF_PIN="${NXF_VER:-25.10.2}"
```

5.17.4 `make_swap.sh`

Creates a swap file. Required: iPHoP peaked at 33.7 GB, above physical RAM.

Show make_swap.sh (106 lines)

```
#!/usr/bin/env bash
set -euo pipefail

# ------------------------------------------------------------------------
# Create a swap file so large single-process allocations are paged instead of
# OOM-killed.
#
# Why this exists: iPHoP's RaFAH stage runs a random forest in R that peaks
# around 19-20 GB. This machine has 31 GB total, ~20 GB actually available once
# the desktop is running, and NO swap - so the kernel kills R outright rather
# than paging:
#     Out of memory: Killed process 2755733 (R) anon-rss:20469616kB
# Swap is not "extra RAM" and will not make anything faster; it is the headroom
# that lets a short memory spike survive instead of being killed.
#
# Run with:  sudo bash make_swap.sh
# Undo with: sudo swapoff /mnt/omics/swapfile && sudo rm /mnt/omics/swapfile
#            (and delete the /etc/fstab line it adds)
# ------------------------------------------------------------------------

SWAPFILE="${SWAPFILE:-/mnt/omics/swapfile}"
SWAPSIZE_GB="${SWAPSIZE_GB:-32}"
SWAPPINESS="${SWAPPINESS:-10}"

if [[ ${EUID} -ne 0 ]]; then
    echo "ERROR: must run as root.  Try:  sudo bash $0" >&2
    exit 1
fi

# --- already done? ----------------------------------------------------------
if swapon --show --noheadings 2>/dev/null | grep -q .; then
    echo "Swap is already active:"
    swapon --show
    echo
    echo "Nothing to do. To resize, swapoff and remove the existing swap first."
    exit 0
fi

if [[ -e "${SWAPFILE}" ]]; then
    echo "ERROR: ${SWAPFILE} already exists but is not active swap." >&2
    echo "Inspect it and remove it yourself before re-running." >&2
    exit 1
fi

# --- sanity checks ----------------------------------------------------------
TARGET_DIR="$(dirname "${SWAPFILE}")"
[[ -d "${TARGET_DIR}" ]] || { echo "ERROR: ${TARGET_DIR} does not exist." >&2; exit 1; }

# Swap files need a real local filesystem. ext4 is fine; network and
# copy-on-write filesystems either refuse or need extra handling.
FSTYPE="$(findmnt -no FSTYPE --target "${TARGET_DIR}")"
```

5.17.5 `quick_test_run.sh`

Section 5.12.6: the quick test run against the pipeline's bundled dataset.

Show quick_test_run.sh (47 lines)

```
nextflow run deng-lab/viroprofiler \
    -r main -profile singularity \
    -c local.config \
    -work-dir work \
    --mode setup \
    --db /mnt/omics/viromics/01_viroprofiler/db \
    --max_cpus 10 --max_memory 24.GB --max_time 48.h


conda activate nextflow_viroprofiler_env
cd /mnt/omics/viromics/01_viroprofiler

# update the pipeline
nextflow pull deng-lab/viroprofiler

# run test
nextflow run deng-lab/viroprofiler -r main \
    -profile singularity,test \
    --db /mnt/omics/viromics/01_viroprofiler/db \
    -c /mnt/omics/viromics/01_viroprofiler/local.config


# test run 2 after fixing dramv and iphop
nextflow run deng-lab/viroprofiler -r main \
    -profile singularity,test \
    --db /mnt/omics/viromics/01_viroprofiler/db \
    -c /mnt/omics/viromics/01_viroprofiler/local.config \
    -resume

# run from cache
cd /mnt/omics/viromics/01_viroprofiler

NXF_VER=25.10.2 nextflow run deng-lab/viroprofiler -r main \
    -profile singularity,test \
    --db /mnt/omics/viromics/01_viroprofiler/db \
    -c /mnt/omics/viromics/01_viroprofiler/local.config \
    -resume 109329a5-d7ff-4bc5-8ba6-8bc111d0634c



#DRAM-v run without dbCAN3
cd /mnt/omics/viromics/01_viroprofiler
NXF_VER=25.10.2 nextflow run deng-lab/viroprofiler -r main \
    -profile singularity,test \
    --db /mnt/omics/viromics/01_viroprofiler/db \
    -c /mnt/omics/viromics/01_viroprofiler/local.config \
    -resume 109329a5-d7ff-4bc5-8ba6-8bc111d0634c
```

5.17.6 `full_analysis_run.sh`

Section 5.12.7: the full analysis run on two public ENA viromes.

Show full_analysis_run.sh (114 lines)

```
#!/usr/bin/env bash
set -euo pipefail

# ============================================================================
# ViroProfiler - demo run on two public viromes
# ============================================================================
#   bash run_test.sh
#
# Downloads two VLP-enriched viromes from ENA study PRJDB10879 and runs the full
# pipeline. Resources are set in local.config, which detects your machine and
# assigns every process explicitly.
#
#   DRR270513   2,406,326 read pairs   ~475 MB
#   DRR270515   2,969,901 read pairs   ~569 MB
#
# WHY THESE, AND NOT SOMETHING SMALLER
# Two earlier attempts failed, each after wasting hours:
#
#   1. ERR14747893 (200k pairs) was almost entirely adapter. fastp discarded
#      100% of it as too_short and SPAdes died on an empty input:
#          == Error ==  file is empty: ERR14747893_1.fastp.fastq.gz
#
#   2. ERR15117916 + ERR15117121 (296k / 488k pairs) had GOOD reads - 97-98%
#      survived fastp - but were far too shallow to assemble. Longest contigs
#      672 bp and 1,668 bp, ZERO reached the 3000 bp threshold, and CheckV then
#      failed on an empty FASTA.
#
# Good reads are not enough; a viral pipeline needs assembly depth. DRR270513
# was assembly-checked before being adopted:
#      77,316 contigs | longest 41,322 bp | 900 >= 3000 bp | 97 >= 10 kb
#
# If you substitute your own accessions, check BOTH read survival and assembly
# length distribution first - see GUIDE.md @sec-dram-database-dead-dbcan-downloads.
#
# RUNTIME: expect roughly 12-24 h. vConTACT2 and iPHoP dominate, and both scale
# with the number of viral contigs found, not with input file size.
# ============================================================================

VP_HOME="${VP_HOME:-$(pwd)}"
DATA_DIR="${VP_HOME}/data/test"
OUT_DIR="${VP_HOME}/results_test"
SHEET="${VP_HOME}/samplesheet_test.csv"
WORK="${VP_HOME}/work_test"
NXF_PIN="${NXF_VER:-25.10.2}"

ENA="https://ftp.sra.ebi.ac.uk/vol1/fastq"
declare -A SAMPLES=(
    [DRR270513]="${ENA}/DRR270/DRR270513"
    [DRR270515]="${ENA}/DRR270/DRR270515"
)
```

6 Complete Viromics Analysis Pipeline

This chapter is the heart of the book: the **core bioinformatics workflow** that turns raw sequencing reads into report-ready viral tables. Think of it as an assembly line. Raw FASTQ enters at one end, and at the other end you get viral genomes with quality scores, taxonomy, functions, abundance, host predictions, and a phylogenetic context. Every station on that line does one job and hands its product to the next.

This chapter uses paired-end reads named:

```
samples_R1.fastq.gz
samples_R2.fastq.gz
```

The workflow is **assembly-based viral mining**. It is suitable for Illumina paired-end DNA virome data and shotgun metagenomes where viral contigs are recovered after assembly. The commands assume you already installed the tools and downloaded the databases (VirSorter2, geNomad, CheckV, DRAM-v, eggNOG, iPHoP, PhaBOX) in the environment-setup chapter.

Learning objectives — by the end of this chapter you will be able to:

- run quality control, trimming, and *de novo* assembly on paired-end virome reads;
- identify viral contigs with VirSorter2 and geNomad and assess their quality with CheckV;
- dereplicate viral genomes into vOTUs and assign conservative taxonomy;
- annotate viral functions, quantify abundance with mapping and TPM, and predict candidate hosts; and
- chain the whole workflow with a reproducible master script and interpret the results honestly.

> **!** Viromics is evidence integration, not one command
>
> A viral contig becomes trustworthy only when **multiple signals agree**: viral prediction, CheckV quality, hallmark genes, even coverage, taxonomy, and biological context. No single tool gives the final truth. Treat every prediction as a hypothesis until the evidence lines converge.

```
flowchart LR
  A[Raw FASTQ] --> B[FastQC and MultiQC]
  B --> C[fastp or Trimmomatic]
  C --> D[MEGAHIT or metaSPAdes]
  D --> E[QUAST]
  E --> F[VirSorter2 and geNomad]
  F --> G[CheckV]
  G --> H[vOTU clustering]
```

```
  H --> I[Taxonomy]
  H --> J[Functional annotation]
  H --> K[Abundance mapping]
  H --> L[Host prediction]
  H --> M[Phylogeny and diversity]
```

6.1 Step 0: Project variables

Start here every time you open a new terminal. These variables and folders are reused by every later step, so defining them once keeps the commands short and portable.

```
cd ~/viromics_course

export VIROME=$HOME/viromics_course
export VIROME_DB=$VIROME/databases
export THREADS=12
export SAMPLE=samples
```

```
export R1=$VIROME/raw_reads/samples_R1.fastq.gz
export R2=$VIROME/raw_reads/samples_R2.fastq.gz

# Database locations used by downstream tools
export CHECKVDB=$VIROME_DB/checkv-db
export EGGNOG_DATA_DIR=$VIROME_DB/eggnog

mkdir -p raw_reads trimmed_reads qc_reports assemblies assembly_qc \
  viral_identification checkv votus taxonomy annotation abundance \
  host_prediction phylogeny diversity visualization/figures logs scripts reports
```

Confirm the reads exist and inspect their statistics before you start:

```
conda activate viromics-core
ls -lh $R1 $R2
seqkit stats $R1 $R2
```

Estimated resources on 12 threads and 32 GB RAM: Folder creation takes seconds and uses almost no storage.

6.2 Step 1: Raw read QC

6.2.1 FastQC

Item	Detail
Purpose	inspect raw read quality before trimming
Install	`mamba install -c conda-forge -c bioconda fastqc`
Input	`samples_R1.fastq.gz`, `samples_R2.fastq.gz`
Output	HTML and ZIP QC reports
Main output folder	`qc_reports/fastqc_raw/`

```
conda activate viromics-core
mkdir -p qc_reports/fastqc_raw

fastqc -t $THREADS -o qc_reports/fastqc_raw $R1 $R2
```

6.2.2 MultiQC

Item	Detail
Purpose	combine many QC reports into one summary HTML
Install	`mamba install -c conda-forge -c bioconda multiqc`

Item	Detail
Input	FastQC output folder
Output	one `multiqc_report.html`
Main output folder	`qc_reports/multiqc_raw/`

```
mkdir -p qc_reports/multiqc_raw

multiqc qc_reports/fastqc_raw \
  -o qc_reports/multiqc_raw \
  -n ${SAMPLE}_raw_multiqc.html
```

On a machine with a browser, open the report with `firefox qc_reports/multiqc_raw/${SAMPLE}_raw_multiqc.` On a headless server, just list the folder to confirm the report was written.

Estimated resources on 12 threads and 32 GB RAM: 5 to 20 minutes for 5 to 15 million read pairs. Storage under 200 MB for QC reports.

6.3 Step 2: Trimming

6.3.1 fastp

fastp performs quality profiling, adapter trimming, read filtering, and base correction in one fast command (Chen et al. 2018).

Item	Detail
Purpose	trim adapters and low-quality bases
Install	`mamba install -c conda-forge -c bioconda fastp`
Input	raw paired FASTQ
Output	trimmed paired FASTQ, HTML report, JSON report
Main output folder	`trimmed_reads/`
Common issue	overly strict filters can remove too many reads

```
mkdir -p trimmed_reads qc_reports/fastp logs

fastp \
  -i $R1 \
  -I $R2 \
  -o trimmed_reads/${SAMPLE}_R1.fastp.fastq.gz \
  -O trimmed_reads/${SAMPLE}_R2.fastp.fastq.gz \
  --detect_adapter_for_pe \
  --cut_front \
  --cut_tail \
  --cut_window_size 4 \
  --cut_mean_quality 20 \
```

```
  --length_required 50 \
  --thread $THREADS \
  --html qc_reports/fastp/${SAMPLE}_fastp.html \
  --json qc_reports/fastp/${SAMPLE}_fastp.json \
  > logs/${SAMPLE}_fastp.log 2>&1

export CLEAN_R1=$VIROME/trimmed_reads/${SAMPLE}_R1.fastp.fastq.gz
export CLEAN_R2=$VIROME/trimmed_reads/${SAMPLE}_R2.fastp.fastq.gz
```

6.3.2 Trimmomatic alternative

Trimmomatic is a flexible Illumina read trimmer that keeps paired and unpaired outputs separate (Bolger, Lohse, and Usadel 2014). Use **either** fastp **or** Trimmomatic — do not trim twice for the same final pipeline unless you are deliberately comparing methods.

```
ADAPTERS=$(find $CONDA_PREFIX -name "TruSeq3-PE.fa" | head -n 1)

trimmomatic PE \
  -threads $THREADS \
  $R1 $R2 \
  trimmed_reads/${SAMPLE}_R1.trimmomatic.paired.fastq.gz \
  trimmed_reads/${SAMPLE}_R1.trimmomatic.unpaired.fastq.gz \
  trimmed_reads/${SAMPLE}_R2.trimmomatic.paired.fastq.gz \
  trimmed_reads/${SAMPLE}_R2.trimmomatic.unpaired.fastq.gz \
  ILLUMINACLIP:${ADAPTERS}:2:30:10 \
  SLIDINGWINDOW:4:20 \
  MINLEN:50
```

Re-run FastQC and MultiQC on the trimmed reads to confirm adapters are gone and quality improved (Shen et al. 2016):

```
mkdir -p qc_reports/fastqc_trimmed qc_reports/multiqc_trimmed

fastqc -t $THREADS -o qc_reports/fastqc_trimmed $CLEAN_R1 $CLEAN_R2
multiqc qc_reports/fastqc_trimmed -o qc_reports/multiqc_trimmed \
  -n ${SAMPLE}_trimmed_multiqc.html
```

Estimated resources on 12 threads and 32 GB RAM: 10 to 30 minutes for the mini project dataset. Temporary storage may need 2x to 3x the compressed FASTQ size.

6.4 Step 3: De novo assembly

6.4.1 MEGAHIT

MEGAHIT is a fast and memory-efficient assembler optimized for large, complex metagenomic datasets (Li et al. 2015).

Item	Detail
Purpose	assemble trimmed reads into contigs
Install	`mamba install -c conda-forge -c bioconda megahit`
Input	trimmed paired FASTQ
Output	assembled contigs FASTA
Main output folder	`assemblies/megahit_samples/`

```
mkdir -p assemblies logs

megahit \
  -1 $CLEAN_R1 \
  -2 $CLEAN_R2 \
  -o assemblies/megahit_${SAMPLE} \
  --min-contig-len 1000 \
  -t $THREADS \
  > logs/${SAMPLE}_megahit.log 2>&1

export CONTIGS=$VIROME/assemblies/megahit_${SAMPLE}/final.contigs.fa
seqkit stats $CONTIGS
grep -c "^>" $CONTIGS
```

6.4.2 metaSPAdes alternative

metaSPAdes is a metagenomic assembler that often produces different contig structures and can yield useful alternative assemblies (Nurk et al. 2017).

```
metaspades.py \
  -1 $CLEAN_R1 \
  -2 $CLEAN_R2 \
  -o assemblies/metaspades_${SAMPLE} \
  -t $THREADS \
  -m 32
```

Teaching recommendation: on a laptop, use MEGAHIT. On a server or HPC, run both and compare. For the rest of this chapter we continue with the MEGAHIT contigs.

Estimated resources on 12 threads and 32 GB RAM: MEGAHIT often takes 30 minutes to 3 hours for moderate teaching datasets and may need 10 to 80 GB temporary storage. metaSPAdes can take several hours and may use most of the 32 GB RAM.

6.5 Step 4: Assembly QC with QUAST

QUAST reports contig count, N50, total length, longest contig, and GC content so you can judge assembly quality (Gurevich et al. 2013).

Item	Detail
Purpose	summarize assembly statistics
Install	`mamba install -c conda-forge -c bioconda quast`
Input	contigs FASTA
Output	HTML report, TSV/TXT statistics
Main output folder	`assembly_qc/quast_megahit/`

```
mkdir -p assembly_qc/quast_megahit

quast.py \
  $CONTIGS \
  -o assembly_qc/quast_megahit \
  -t $THREADS \
  --min-contig 1000

cat assembly_qc/quast_megahit/report.txt
```

Interpretation focuses on contig count, total length, N50, and longest contig. A good viral assembly is **not** defined by N50 alone. Viral recovery depends on read depth, genome complexity, enrichment, and database support.

Estimated resources on 12 threads and 32 GB RAM: 2 to 10 minutes. Storage under 200 MB.

6.6 Step 5: Viral identification

No single viral prediction tool is perfect. In teaching and in practice, use at least two evidence types. The core here is **VirSorter2 + geNomad + CheckV**, with VirFinder and DeepVirFinder as optional score-based comparisons.

6.6.1 VirSorter2

VirSorter2 identifies diverse DNA and RNA viral sequences using multiple viral classifiers and reports a viral score per contig (Guo et al. 2021).

Item	Detail
Purpose	identify viral candidate contigs
Install	`mamba create -n virsorter2 -c conda-forge -c bioconda virsorter=2`
Input	assembly contigs FASTA
Output	viral candidate FASTA and score tables
Main output folder	`viral_identification/virsorter2/`

```
conda activate virsorter2
mkdir -p viral_identification/virsorter2 logs

virsorter run \
  -i $CONTIGS \
  -w viral_identification/virsorter2 \
  --keep-original-seq \
  --min-length 1000 \
  --min-score 0.5 \
  -j $THREADS \
  all \
  > logs/${SAMPLE}_virsorter2.log 2>&1

seqkit stats viral_identification/virsorter2/final-viral-combined.fa
head viral_identification/virsorter2/final-viral-score.tsv
```

6.6.2 geNomad

geNomad identifies viruses and plasmids from nucleotide sequences, assigns taxonomy following ICTV releases, and annotates genes in a single `end-to-end` command (Camargo et al. 2024).

Item	Detail
Purpose	identify viruses/plasmids and assign taxonomy
Install	`mamba create -n genomad -c conda-forge -c bioconda genomad`
Input	contigs FASTA
Output	virus/plasmid predictions, taxonomy, gene annotation
Main output folder	`viral_identification/genomad/`

```
conda activate genomad
mkdir -p viral_identification/genomad logs

genomad end-to-end \
  --threads $THREADS \
  $CONTIGS \
  viral_identification/genomad \
  $VIROME_DB/genomad/genomad_db \
  > logs/${SAMPLE}_genomad.log 2>&1

find viral_identification/genomad -type f | grep -E "virus|summary|taxonomy|faa|fna" | head -n
```

6.6.3 VirFinder and DeepVirFinder

These tools add k-mer/signature-based and deep-learning score evidence, useful especially for short contigs (Ren et al. 2017, 2020). They are excellent for teaching score-based prediction, but modern publication workflows should prioritize VirSorter2 + geNomad + CheckV. The VirFinder R script already exists in the book scripts.

```
conda activate virfinder
mkdir -p viral_identification/virfinder
Rscript ../scripts/run_virfinder.R $CONTIGS \
  viral_identification/virfinder/${SAMPLE}_virfinder.tsv
```

```
conda activate deepvirfinder
dvf.py -i $CONTIGS -o viral_identification/deepvirfinder -l 1000 -c $THREADS
```

6.6.4 Combine viral candidates

For a practical first workflow, take the VirSorter2 candidates forward to CheckV. If a geNomad virus FASTA is available, you can combine and deduplicate later.

```
mkdir -p viral_identification/combined

cp viral_identification/virsorter2/final-viral-combined.fa \
   viral_identification/combined/${SAMPLE}_viral_candidates.fa

seqkit stats viral_identification/combined/${SAMPLE}_viral_candidates.fa
```

Estimated resources on 12 threads and 32 GB RAM: VirSorter2 and geNomad can take 30 minutes to several hours depending on contig count and database speed. Reserve 20 to 100 GB for outputs and databases.

6.7 Step 6: CheckV quality assessment

CheckV removes host contamination in proviruses, estimates completeness, detects closed genomes, and assigns quality tiers (Nayfach et al. 2021).

> **i** CheckV quality tiers
>
> CheckV assigns each contig to a tier from its estimated **completeness**: **Complete** (a closed genome confirmed by terminal repeats or a reference), **High-quality** (>90%), **Medium-quality** (50–90%), **Low-quality** (<50%), and **Not-determined** (too little evidence to estimate). Crucially, **contamination is deliberately *not* factored into the tier** — host flanks on a provirus are easy to trim, so a high-contamination contig is cleaned rather than downgraded. Here "contamination" means host DNA flanking an integrated provirus, not cross-sample contamination.

Item	Detail
Purpose	assess viral genome quality and completeness
Install	`mamba create -n checkv -c conda-forge -c bioconda checkv`
Input	viral candidate FASTA
Output	`quality_summary.tsv`, `viruses.fna`, `proviruses.fna`, completeness tables
Main output folder	`checkv/checkv_samples/`

```
conda activate checkv
mkdir -p checkv/checkv_${SAMPLE} logs

checkv end_to_end \
  viral_identification/combined/${SAMPLE}_viral_candidates.fa \
  checkv/checkv_${SAMPLE} \
  -t $THREADS \
  -d $CHECKVDB \
  > logs/${SAMPLE}_checkv.log 2>&1

cut -f1-10 checkv/checkv_${SAMPLE}/quality_summary.tsv | column -t | head
```

Build a combined CheckV-cleaned FASTA, then filter to the Medium-quality, High-quality, and Complete contigs.

```
cat checkv/checkv_${SAMPLE}/viruses.fna \
    checkv/checkv_${SAMPLE}/proviruses.fna \
    > checkv/checkv_${SAMPLE}/${SAMPLE}_checkv_combined_viral.fna

awk -F'\t' '
NR==1 {next}
$8=="Medium-quality" || $8=="High-quality" || $8=="Complete" {
  print $1
}' checkv/checkv_${SAMPLE}/quality_summary.tsv \
> checkv/checkv_${SAMPLE}/${SAMPLE}_medium_high_ids.txt

seqkit grep \
  -f checkv/checkv_${SAMPLE}/${SAMPLE}_medium_high_ids.txt \
  checkv/checkv_${SAMPLE}/${SAMPLE}_checkv_combined_viral.fna \
  > checkv/checkv_${SAMPLE}/${SAMPLE}_medium_high_viral_contigs.fna

export VIRAL_FASTA=$VIROME/checkv/checkv_${SAMPLE}/${SAMPLE}_medium_high_viral_contigs.fna
seqkit stats $VIRAL_FASTA
```

> **Warning**
>
> If `VIRAL_FASTA` is empty, fall back to all CheckV combined contigs so you can keep practicing, and report clearly that strict quality filtering produced no medium or better genomes.

```
export VIRAL_FASTA=$VIROME/checkv/checkv_${SAMPLE}/${SAMPLE}_checkv_combined_viral.fna
```

Estimated resources on 12 threads and 32 GB RAM: 10 minutes to 2 hours depending on viral candidate count. Storage often under 10 GB beyond the database.

6.8 Step 7: vOTU clustering

A common species-level operational definition for dsDNA viral populations uses **95% ANI over 85% alignment fraction (AF)** of the shorter sequence — the MIUViG community standard (Roux et al. 2019). The alignment-fraction cutoff matters: 95% ANI *alone*, with no AF constraint, can merge unrelated genomes that happen to share one highly conserved region. Always report your exact thresholds.

6.8.1 vClust

vClust calculates ANI between viral genomes and clusters them into vOTUs. Pass `--qcov 0.85` alongside `--ani 0.95` to enforce the 85% alignment fraction the MIUViG standard requires — the `--ani 0.95` quick-start on its own does not (Zielezinski et al. 2025).

Item	Detail
Purpose	ANI-aware clustering of viral genomes into vOTUs
Install	`mamba create -n vclust -c conda-forge -c bioconda vclust`
Input	viral FASTA
Output	ANI table and cluster table
Main output folder	`votus/vclust_samples/`

```
conda activate vclust
mkdir -p votus/vclust_${SAMPLE}

vclust prefilter -i $VIRAL_FASTA -o votus/vclust_${SAMPLE}/prefilter.tsv
vclust align -i $VIRAL_FASTA -o votus/vclust_${SAMPLE}/ani.tsv \
  --filter votus/vclust_${SAMPLE}/prefilter.tsv
vclust cluster -i votus/vclust_${SAMPLE}/ani.tsv \
  -o votus/vclust_${SAMPLE}/clusters.tsv \
  --ids votus/vclust_${SAMPLE}/ani.ids.tsv \
  --algorithm leiden \
  --metric ani \
  --ani 0.95 \
  --qcov 0.85
```

6.8.2 CD-HIT-EST

CD-HIT-EST is a fast nucleotide clustering tool, handy as a simple teaching dereplication method (Fu et al. 2012).

Item	Detail
Purpose	fast nucleotide dereplication into representatives
Install	`mamba install -c conda-forge -c bioconda cd-hit`
Input	viral FASTA
Output	representative FASTA and `.clstr` cluster file
Main output folder	`votus/cdhit_samples/`

```
conda activate viromics-core
mkdir -p votus/cdhit_${SAMPLE}

cd-hit-est \
  -i $VIRAL_FASTA \
  -o votus/cdhit_${SAMPLE}/${SAMPLE}_votus_95.fa \
  -c 0.95 \
  -aS 0.85 \
  -G 0 \
  -g 1 \
  -T $THREADS \
  -M 0

export VOTU_FASTA=$VIROME/votus/cdhit_${SAMPLE}/${SAMPLE}_votus_95.fa
seqkit stats $VOTU_FASTA
```

Use vClust for ANI-aware viral clustering and CD-HIT-EST as a simpler alternative. For publication, report the exact ANI and coverage criteria you used.

Estimated resources on 12 threads and 32 GB RAM: Small datasets finish in minutes. Thousands of viral genomes can take hours and need tens of GB.

6.9 Step 8: Taxonomy

Combine geNomad taxonomy, vConTACT2 gene-sharing networks, and PhaBOX/PhaGCN outputs within the ICTV framework (International Committee on Taxonomy of Viruses 2026). If tools disagree, report the most conservative rank. Do not overstate uncertain ranks.

6.9.1 geNomad taxonomy

Item	Detail
Purpose	assign virus taxonomy with a marker-based workflow
Install	`mamba create -n genomad -c conda-forge -c bioconda genomad`
Input	vOTU FASTA
Output	taxonomy tables, virus summary
Main output folder	`taxonomy/genomad_votu/`

```
conda activate genomad
mkdir -p taxonomy/genomad_votu logs

genomad end-to-end \
  --threads $THREADS \
  $VOTU_FASTA \
  taxonomy/genomad_votu \
  $VIROME_DB/genomad/genomad_db \
  > logs/${SAMPLE}_genomad_votu_taxonomy.log 2>&1
```

6.9.2 Prodigal proteins for vConTACT2

Prodigal predicts protein-coding genes and supports a metagenomic mode (Hyatt et al. 2010). vConTACT2 needs the protein FASTA plus a gene-to-genome CSV map.

Item	Detail
Purpose	predict ORFs/proteins from vOTU contigs
Install	`mamba install -c conda-forge -c bioconda prodigal`
Input	vOTU FASTA
Output	proteins `.faa`, genes `.fna`, annotation `.gff`
Main output folder	`annotation/prodigal/`

```
conda activate viromics-core
mkdir -p annotation/prodigal taxonomy/vcontact2

prodigal \
  -i $VOTU_FASTA \
  -a annotation/prodigal/${SAMPLE}_votus.faa \
  -d annotation/prodigal/${SAMPLE}_votus.genes.fna \
  -o annotation/prodigal/${SAMPLE}_votus.gff \
  -f gff \
  -p meta
```

Build the gene-to-genome table from the Prodigal headers:

```
grep "^>" annotation/prodigal/${SAMPLE}_votus.faa \
| sed 's/^>//' \
| awk 'BEGIN{FS=" "; OFS=","} {
    protein=$1;
    contig=$1;
    sub(/_[0-9]+$/, "", contig);
    print protein, contig, "unknown"
}' > taxonomy/vcontact2/g2g.tmp.csv

echo "protein_id,contig_id,keywords" \
| cat - taxonomy/vcontact2/g2g.tmp.csv \
> taxonomy/vcontact2/g2g.csv
```

6.9.3 vConTACT2

vConTACT2 builds a gene-sharing network and clusters viral genomes into taxonomy-informative viral clusters (VCs).

Item	Detail
Purpose	gene-sharing network taxonomy for prokaryotic viruses
Install	`mamba create -n vcontact2 -c conda-forge -c bioconda vcontact2 mcl blast diamond prodigal`
Input	protein FASTA + gene-to-genome map
Output	viral clusters and genome-by-genome overview
Main output folder	`taxonomy/vcontact2/vcontact_out/`

```
conda activate vcontact2

vcontact2 \
  --raw-proteins annotation/prodigal/${SAMPLE}_votus.faa \
  --rel-mode Diamond \
  --proteins-fp taxonomy/vcontact2/g2g.csv \
  --db ProkaryoticViralRefSeq201-Merged \
  --pcs-mode MCL \
  --vcs-mode ClusterONE \
  --output-dir taxonomy/vcontact2/vcontact_out \
  > logs/${SAMPLE}_vcontact2.log 2>&1
```

6.9.3.1 Reading the vConTACT2 network

vConTACT2 places your viral genomes and the reference genomes into a network where nodes are genomes and edges are weighted by shared protein clusters. Two files carry the interpretation:

```
head taxonomy/vcontact2/vcontact_out/genome_by_genome_overview.csv
head taxonomy/vcontact2/vcontact_out/viral_cluster_overview.csv
```

- **`genome_by_genome_overview.csv`** lists each genome, its assigned VC, and the taxonomy of the reference genomes sharing that cluster. When your vOTU lands in a VC that also contains a well-classified reference, that reference lineage is your best taxonomy hypothesis.
- **`viral_cluster_overview.csv`** summarizes each VC. A VC containing **only** your genomes and no references is a signal of **novelty** — an honest result to report as an unclassified viral cluster rather than forcing a species name.
- Genomes flagged as **"Outlier"** or **"Overlap"** did not join a clean cluster; treat their taxonomy as unresolved.

6.9.4 PhaBOX / PhaGCN

PhaBOX2 integrates several phage modules (PhaMer, PhaGCN, PhaTYP, CHERRY/HostG, PhaVIP) for identification, taxonomy, lifestyle, and host prediction; PhaGCN is its taxonomy classifier (Shang et al. 2026).

Item	Detail
Purpose	phage taxonomy, lifestyle, and host modules
Install	`mamba create -n phabox -c conda-forge -c bioconda phabox`
Input	vOTU FASTA
Output	phage taxonomy, lifestyle, host, annotation tables
Main output folder	`taxonomy/phabox/`

```
conda activate phabox
mkdir -p taxonomy/phabox logs

phabox2 \
  --task end_to_end \
  --contigs $VOTU_FASTA \
  --outpth taxonomy/phabox \
  --dbdir $VIROME_DB/phabox \
  --threads $THREADS \
  > logs/${SAMPLE}_phabox.log 2>&1
```

The current release ships the `phabox2` binary and takes `--contigs` (the older `phabox --input` form is gone). PhaBOX commands still vary by version — if the syntax above fails, check `phabox2 --help`.

Estimated resources on 12 threads and 32 GB RAM: geNomad taxonomy can take 30 minutes to hours. vConTACT2 is memory sensitive when many genomes are included. Reserve 20 to 100 GB.

6.10 Step 9: Functional annotation

6.10.1 eggNOG-mapper

eggNOG-mapper annotates predicted proteins using precomputed orthologous groups and phylogenies.

Item	Detail
Purpose	functional annotation of predicted proteins via orthology
Install	`mamba create -n eggnog -c conda-forge -c bioconda eggnog-mapper`
Input	protein FASTA
Output	functional annotation table
Main output folder	`annotation/eggnog/`

```
conda activate eggnog
mkdir -p annotation/eggnog logs

emapper.py \
  -i annotation/prodigal/${SAMPLE}_votus.faa \
  --itype proteins \
  -m diamond \
  --cpu $THREADS \
  --data_dir $EGGNOG_DATA_DIR \
  -o ${SAMPLE}_eggnog \
  --output_dir annotation/eggnog \
  > logs/${SAMPLE}_eggnog.log 2>&1

head annotation/eggnog/${SAMPLE}_eggnog.emapper.annotations
```

6.10.2 VIBRANT

VIBRANT automates viral recovery, annotation, and curation from metagenomic assemblies, and is well suited to interpreting viral protein functions (Kieft, Zhou, and Anantharaman 2020).

Item	Detail
Purpose	viral identification, annotation, and curation
Install	`mamba create -n vibrant -c conda-forge -c bioconda vibrant`
Input	viral/vOTU FASTA
Output	viral annotations, protein predictions, summaries
Main output folder	`annotation/vibrant/`

```
conda activate vibrant
mkdir -p annotation/vibrant logs

VIBRANT_run.py \
  -i $VOTU_FASTA \
  -folder annotation/vibrant \
  -t $THREADS \
  -virome \
  -d $VIROME_DB/vibrant/databases \
  > logs/${SAMPLE}_vibrant.log 2>&1
```

If the database path differs, locate it with `find $VIROME_DB/vibrant -maxdepth 3 -type d`.

6.10.3 DRAM-v

DRAM-v annotates viral contigs and helps identify **auxiliary metabolic genes (AMGs)** — host metabolism genes carried by phages (Shaffer et al. 2020). Its input is a special VirSorter2 preparation, so the workflow is three stages: prep, annotate, distill.

Item	Detail
Purpose	viral function annotation and AMG interpretation
Install	`mamba env create -f DRAM_environment.yaml -n dramv`
Input	viral FASTA + VirSorter2 DRAM-v prep table
Output	annotations, GFF, genes, AMG summary
Main output folder	`annotation/dramv/`

First, prepare DRAM-v-compatible output with VirSorter2 in `--prep-for-dramv` mode:

```
conda activate virsorter2
mkdir -p annotation/dramv/vs2_for_dramv logs

virsorter run \
  --seqname-suffix-off \
  --viral-gene-enrich-off \
  --provirus-off \
  --prep-for-dramv \
  -i $VOTU_FASTA \
  -w annotation/dramv/vs2_for_dramv \
  --min-length 1000 \
  --min-score 0.5 \
  -j $THREADS \
  all \
  > logs/${SAMPLE}_vs2_for_dramv.log 2>&1
```

Annotate, then distill the AMG summary:

```
conda activate dramv
mkdir -p annotation/dramv/dramv_annotate

DRAM-v.py annotate \
  -i annotation/dramv/vs2_for_dramv/for-dramv/final-viral-combined-for-dramv.fa \
  -v annotation/dramv/vs2_for_dramv/for-dramv/viral-affi-contigs-for-dramv.tab \
  -o annotation/dramv/dramv_annotate \
  --skip_trnascan \
  --threads $THREADS \
  --min_contig_size 1000

DRAM-v.py distill \
  -i annotation/dramv/dramv_annotate/annotations.tsv \
  -o annotation/dramv/dramv_distill \
  --max_auxiliary_score 3

head annotation/dramv/dramv_distill/amg_summary.tsv
```

Estimated resources on 12 threads and 32 GB RAM: eggNOG annotation can take minutes to hours depending on protein count. DRAM-v and VIBRANT are heavy because of database searches. Reserve 50 GB or more when running full annotation databases.

6.11 Step 10: Abundance and TPM

Assembly answers *which viral contigs exist*. Mapping answers *how abundant each contig is in the reads*. For a single paired-end dataset, map the cleaned reads back to the vOTUs.

6.11.1 Bowtie2

Bowtie2 aligns reads to the vOTU reference; SAMtools sorts and indexes the alignment (Langmead and Salzberg 2012; Danecek et al. 2021).

Item	Detail
Purpose	align reads to vOTUs
Install	`mamba install -c conda-forge -c bioconda bowtie2 samtools`
Input	clean FASTQ + vOTU FASTA
Output	sorted, indexed BAM alignment
Main output folder	`abundance/`

```
conda activate viromics-core
mkdir -p abundance/bowtie2_index abundance/logs

bowtie2-build $VOTU_FASTA abundance/bowtie2_index/${SAMPLE}_votus
```

```
bowtie2 \
  -x abundance/bowtie2_index/${SAMPLE}_votus \
  -1 $CLEAN_R1 \
  -2 $CLEAN_R2 \
  -p $THREADS \
  --very-sensitive \
  2> abundance/logs/${SAMPLE}_bowtie2_mapping.log \
| samtools view -bS - \
| samtools sort -@ $THREADS -o abundance/${SAMPLE}_vs_votus.sorted.bam

samtools index abundance/${SAMPLE}_vs_votus.sorted.bam
samtools flagstat abundance/${SAMPLE}_vs_votus.sorted.bam \
  > abundance/${SAMPLE}_flagstat.txt
```

6.11.2 CoverM

CoverM calculates coverage and relative abundance from a BAM file or directly from reads.

Item	Detail
Purpose	compute contig coverage and abundance
Install	`mamba install -c conda-forge -c bioconda coverm`
Input	BAM, or paired FASTQ + vOTU FASTA
Output	coverage table
Main output folder	`abundance/`

Using the sorted BAM:

```
coverm contig \
  --bam-files abundance/${SAMPLE}_vs_votus.sorted.bam \
  --methods mean covered_fraction count tpm \
  --threads $THREADS \
  > abundance/${SAMPLE}_coverm_contig.tsv
```

CoverM can also map directly from raw reads in one step, without a pre-built BAM:

```
coverm contig \
  --coupled $CLEAN_R1 $CLEAN_R2 \
  --reference $VOTU_FASTA \
  --mapper minimap2-sr \
  --methods mean covered_fraction count tpm \
  --threads $THREADS \
  > abundance/${SAMPLE}_coverm_direct.tsv
```

6.11.3 Manual TPM (conceptual walkthrough)

To understand what CoverM does internally, compute TPM by hand. TPM normalizes for both contig length and sequencing depth:

```
reads mapped to contig
  ÷ contig length in kb        → reads per kilobase (RPK)
  ÷ (sum of all RPK ÷ 1e6)     → per-million scaling
  = TPM
```

Get contig lengths and mapped read counts:

```
seqkit fx2tab -n -l $VOTU_FASTA \
  > abundance/${SAMPLE}_votu_lengths.tsv

samtools idxstats abundance/${SAMPLE}_vs_votus.sorted.bam \
| awk '$1!="*" {print $1"\t"$2"\t"$3}' \
> abundance/${SAMPLE}_mapped_counts.tsv
```

Then apply the RPK/TPM formula with the ready-made script (it reads `abundance/samples_mapped_counts.tsv` and writes `abundance/samples_manual_tpm.tsv`):

```
python calculate_tpm.py \
  abundance/${SAMPLE}_mapped_counts.tsv \
  abundance/${SAMPLE}_manual_tpm.tsv
head abundance/${SAMPLE}_manual_tpm.tsv
```

Estimated resources on 12 threads and 32 GB RAM: Mapping can take 10 minutes to 2 hours. BAM files can be similar in size to or larger than the compressed input FASTQ files.

6.12 Step 11: Host prediction

Host prediction is uncertain. Always report a **candidate host**, never absolute proof of infection. The strongest results combine evidence types: CRISPR spacer matches, sequence composition, iPHoP, WIsH, and taxonomy context.

6.12.1 CRISPR spacer matching with BLASTn

CRISPR spacers record previous encounters between a host and its viruses, so a spacer matching a viral contig is a strong host clue.

Item	Detail
Purpose	match known host CRISPR spacers to viral contigs
Install	`mamba install -c conda-forge -c bioconda blast seqkit`

Item	Detail
Input	host spacer FASTA + viral FASTA
Output	BLAST table of spacer-virus matches
Main output folder	`host_prediction/`

```
conda activate viromics-core
mkdir -p host_prediction

makeblastdb \
  -in $VOTU_FASTA \
  -dbtype nucl \
  -out host_prediction/votu_blast_db

blastn \
  -query host_prediction/example_host_spacers.fa \
  -db host_prediction/votu_blast_db \
  -task blastn-short \
  -perc_identity 95 \
  -qcov_hsp_perc 95 \
  -outfmt "6 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitsco
  -num_threads $THREADS \
  -out host_prediction/${SAMPLE}_crispr_spacer_hits.tsv
```

6.12.2 iPHoP

iPHoP is an integrated machine-learning framework that predicts the host genus for cultivated and uncultivated phages and archaeal viruses (Roux et al. 2023).

Item	Detail
Purpose	integrated host-genus prediction
Install	`mamba create -n iphop -c conda-forge -c bioconda iphop`
Input	viral FASTA
Output	host prediction tables
Main output folder	`host_prediction/iphop/`

```
conda activate iphop
mkdir -p host_prediction/iphop logs

# iPHoP unpacks into a versioned subfolder (e.g. Aug_2023_pub_rw/); point at that, not its pare
IPHOP_DB=$(find $VIROME_DB/iphop -maxdepth 1 -type d -name "*_pub_*" | head -n1)

iphop predict \
  --fa_file $VOTU_FASTA \
  --db_dir $IPHOP_DB \
```

```
  --out_dir host_prediction/iphop \
  --num_threads $THREADS \
  > logs/${SAMPLE}_iphop.log 2>&1
```

6.12.3 WIsH

WIsH predicts prokaryotic hosts of phage contigs using Markov models trained on candidate host genomes (Galiez et al. 2017). It expects **one genome per file**, so split the vOTUs first.

Item	Detail
Purpose	Markov-model host prediction from candidate genomes
Install	compile from GitHub, or use your `wish` env
Input	host genome FASTA directory + viral genome FASTA directory
Output	likelihood matrix and host prediction scores
Main output folder	`host_prediction/wish/`

Split each viral genome into its own file:

```
conda activate viromics-core
mkdir -p host_prediction/wish/viral_genomes host_prediction/wish/host_genomes

seqkit split -i -O host_prediction/wish/viral_genomes $VOTU_FASTA
```

Place relevant candidate host genomes (one per file) in `host_prediction/wish/host_genomes/`, for example `Escherichia_coli.fna`, `Pseudomonas_aeruginosa.fna`, `Bacillus_subtilis.fna`. Then build the host models and predict:

```
conda activate wish
mkdir -p host_prediction/wish/models host_prediction/wish/results

WIsH -c build \
  -g host_prediction/wish/host_genomes \
  -m host_prediction/wish/models \
  -t $THREADS

WIsH -c predict \
  -g host_prediction/wish/viral_genomes \
  -m host_prediction/wish/models \
  -r host_prediction/wish/results \
  -t $THREADS
```

WIsH is only meaningful with relevant candidate host genomes: use gut bacterial genomes for gut viromes, soil/rhizosphere MAGs for soil viromes, and marine microbial genomes for marine viromes.

Estimated resources on 12 threads and 32 GB RAM: CRISPR BLAST is usually quick if spacer files are small. iPHoP runtime varies widely and can require large database storage, often more than 100 GB.

6.13 Step 12: Phylogenetics and diversity

MAFFT aligns marker genes (Katoh and Standley 2013). IQ-TREE 2 infers maximum-likelihood trees with model selection and ultrafast bootstrap (Minh et al. 2020).

6.13.1 MAFFT and IQ-TREE

Item	Detail
Purpose	align a marker family (MAFFT) and infer a tree (IQ-TREE)
Install	`mamba install -c conda-forge -c bioconda mafft iqtree`
Input	homologous protein or nucleotide FASTA
Output	aligned FASTA; `.treefile`, model report, log
Main output folder	`phylogeny/`

```
conda activate viromics-core
mkdir -p phylogeny

seqkit head -n 50 annotation/prodigal/${SAMPLE}_votus.faa \
  > phylogeny/${SAMPLE}_demo_50_proteins.faa

mafft --auto --thread $THREADS \
  phylogeny/${SAMPLE}_demo_50_proteins.faa \
  > phylogeny/${SAMPLE}_demo_50_proteins.aligned.faa

iqtree \
  -s phylogeny/${SAMPLE}_demo_50_proteins.aligned.faa \
  -m MFP \
  -B 1000 \
  -T AUTO \
  --prefix phylogeny/${SAMPLE}_demo_tree
```

> **Warning**
>
> Do not build a biological tree from unrelated proteins. The demo above is only a mechanics exercise. For real studies, extract **one homologous marker family** — terminase large subunit, major capsid protein, portal protein, or RdRp for RNA viruses — before aligning.

6.13.2 Diversity and richness

Once you have per-vOTU abundances, ecological summaries follow. Copy the TPM table into the diversity folder as a single-sample abundance matrix:

```
mkdir -p diversity
cp abundance/${SAMPLE}_manual_tpm.tsv diversity/${SAMPLE}_votu_abundance.tsv
```

For several samples you would build a wide matrix (`vOTU` rows, one `*_TPM` column per sample). A simple **richness** count — how many vOTUs are detected in a sample — comes straight from `awk`:

```
awk -F'\t' 'NR>1 && $5>0 {count++} END {print "Detected_vOTUs\t"count}' \
  abundance/${SAMPLE}_manual_tpm.tsv \
  > diversity/${SAMPLE}_richness.tsv

cat diversity/${SAMPLE}_richness.tsv
```

Richness is the starting point for alpha diversity; with multiple samples you can extend this matrix into Shannon diversity and between-sample (beta) comparisons.

Estimated resources on 12 threads and 32 GB RAM: Small marker alignments finish in minutes. Hundreds to thousands of sequences with bootstrapping can take hours.

6.14 Master pipeline script

Running every step by hand is great for learning but tedious to repeat. Two ready-made scripts chain the core steps for you. Both use `conda run -n <env>` so each tool runs in its own environment without manual `conda activate` calls, which makes them safe to run non-interactively.

`run_phase4_main_pipeline.sh` runs QC, trimming, MEGAHIT assembly, QUAST, VirSorter2, CheckV, CD-HIT vOTUs, and Bowtie2/CoverM abundance end to end, writing logs and a summary as it goes. `create_report_summary.sh` collects the key `seqkit stats` and output paths into a single human-readable report.

```
chmod +x scripts/run_phase4_main_pipeline.sh
bash scripts/run_phase4_main_pipeline.sh

bash scripts/create_report_summary.sh "$VIROME" "$SAMPLE"
cat reports/${SAMPLE}_viromics_summary.txt
```

6.15 Hands-on exercises

Exercise 1: reads before and after trimming

Run FastQC and fastp on `samples_R1.fastq.gz` / `samples_R2.fastq.gz`, then report the read count before and after trimming.
One solution:

```
cd ~/viromics_course
conda activate viromics-core
export SAMPLE=samples THREADS=12
export R1=raw_reads/samples_R1.fastq.gz R2=raw_reads/samples_R2.fastq.gz
mkdir -p qc_reports/fastqc_raw qc_reports/fastp trimmed_reads

fastqc -t $THREADS -o qc_reports/fastqc_raw $R1 $R2
fastp -i $R1 -I $R2 \
  -o trimmed_reads/${SAMPLE}_R1.fastp.fastq.gz \
  -O trimmed_reads/${SAMPLE}_R2.fastp.fastq.gz \
  --detect_adapter_for_pe --thread $THREADS \
  --html qc_reports/fastp/${SAMPLE}_fastp.html \
  --json qc_reports/fastp/${SAMPLE}_fastp.json

seqkit stats $R1 $R2 \
  trimmed_reads/${SAMPLE}_R1.fastp.fastq.gz \
  trimmed_reads/${SAMPLE}_R2.fastp.fastq.gz
```

Exercise 2: count contigs longer than 5 kb

Assemble the trimmed reads with MEGAHIT and count contigs longer than 5 kb.
One solution:

```
megahit \
  -1 trimmed_reads/samples_R1.fastp.fastq.gz \
  -2 trimmed_reads/samples_R2.fastp.fastq.gz \
  -o assemblies/megahit_samples \
  --min-contig-len 1000 -t 12

seqkit seq -m 5000 assemblies/megahit_samples/final.contigs.fa \
  > assemblies/megahit_samples/contigs_gt5kb.fa
grep -c "^>" assemblies/megahit_samples/contigs_gt5kb.fa
```

Common mistakes

- **Treating every viral prediction as a true virus.** Prediction tools produce *candidates*. Confirm with CheckV quality, hallmark genes, contamination checks, and agreement across tools.

- **Forgetting to map reads back to viral contigs.** Assembly gives presence; mapping gives abundance and coverage. You need both to interpret a virome.
- **Using taxonomy too confidently.** Many viral contigs are novel. Report the most reliable rank and use terms like *putative*, *candidate*, and *unclassified viral contig*.
- **Building trees from unrelated proteins.** Phylogenies require one homologous marker family, not a mix of all predicted proteins.

6.16 Key takeaways

- The pipeline is an assembly-based assembly line: QC and trimming, *de novo* assembly with MEGAHIT (Li et al. 2015), viral identification, CheckV quality (Nayfach et al. 2021), vOTU clustering, then the downstream taxonomy, function, abundance, host, and phylogeny branches.
- No single predictor is trustworthy alone; treat VirSorter2 (Guo et al. 2021) plus geNomad (Camargo et al. 2024) plus CheckV as the core, and require multiple evidence lines to agree before calling a contig viral.
- Assembly tells you which viral contigs exist; mapping reads back with Bowtie2/CoverM tells you how abundant they are, so both presence and TPM abundance are needed to interpret a virome.
- Report taxonomy conservatively and label novel sequences as putative or unclassified, since reference databases remain incomplete for viruses.
- Chain the steps with a reproducible master script that runs each tool via `conda run -n <env>`, and build phylogenies only from a single homologous marker family (Minh et al. 2020).

6.17 Further reading

- geNomad end-to-end virus/plasmid identification, taxonomy, and gene annotation (Camargo et al. 2024); project site: https://github.com/apcamargo/genomad.
- CheckV for interpreting completeness, contamination, and quality tiers of assembled viral genomes (Nayfach et al. 2021).
- iPHoP for integrated host-genus prediction and how to read its confidence scores (Roux et al. 2023).
- DRAM-v for auxiliary metabolic gene annotation of viral contigs (Shaffer et al. 2020).

6.18 Chapter figure

Generate this figure

Save as: `images/ch04-pipeline-overview.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean horizontal workflow infographic of a complete viromics analysis

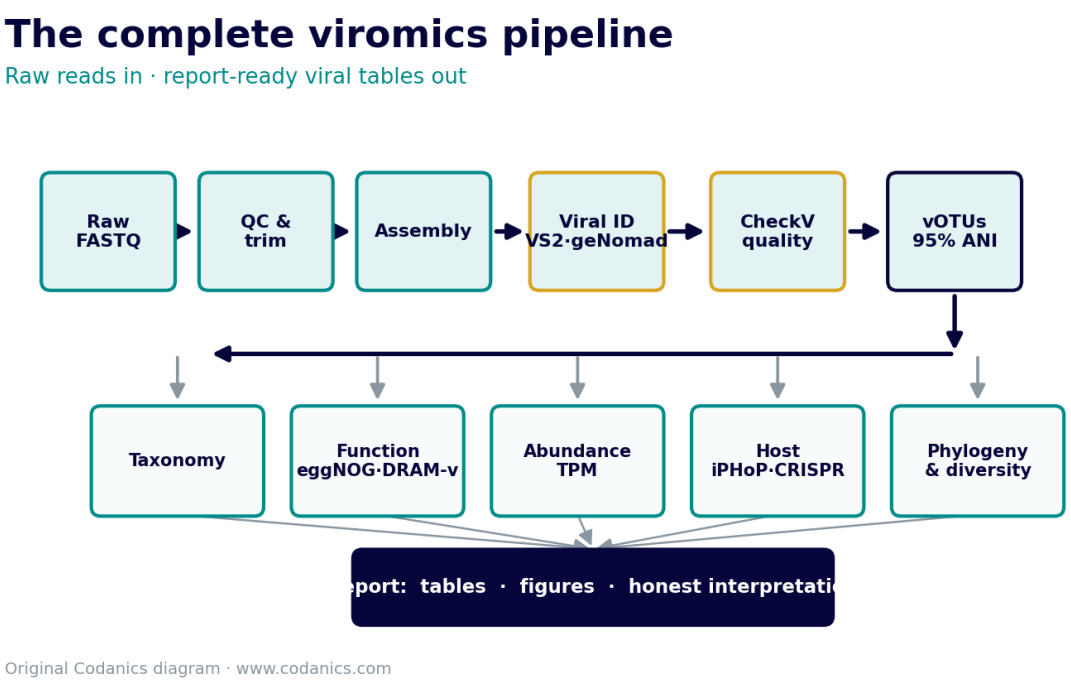


Figure 6.1: The complete assembly-based viromics pipeline, from paired-end FASTQ through QC, assembly, viral identification, CheckV, vOTU clustering, and the downstream taxonomy, function, abundance, host, and phylogeny branches to a final report.

pipeline. From left to right, show labeled stages connected by arrows: paired-end FASTQ reads, quality control (FastQC/MultiQC), trimming (fastp), de novo assembly (MEGAHIT), assembly QC (QUAST), viral identification (VirSorter2 and geNomad), quality assessment (CheckV), and vOTU clustering. After vOTU clustering, fan the flow into five parallel downstream branches, each in its own rounded box: taxonomy, functional annotation, abundance mapping, host prediction, and phylogeny/diversity. Converge all branches into a final "report-ready tables" box on the right. Use small Linux-terminal icons on the compute steps, teal and navy Codanics branding, thin connecting arrows, and clear stage labels. No photorealism.

6.19 Quiz: Complete Pipeline

Q1. Which step should happen before assembly?

A. read QC and trimming B. vConTACT2 C. host prediction D. tree building

Note

Answer: A. Low-quality reads and adapters should be handled before assembly.

Q2. What does CheckV estimate?

A. viral quality and completeness B. adapter sequence only C. CPU temperature D. sequencer model

> **Note**
>
> **Answer:** A. CheckV is used for viral genome quality assessment.

Q3. What does mapping reads back to vOTUs estimate?

A. abundance and coverage B. Baltimore group C. library kit price D. host genome size

> **Note**
>
> **Answer:** A. Coverage and TPM are calculated from mapped reads.

Q4. Why should taxonomy be conservative?

A. many viruses are novel B. all viruses are known C. taxonomy is never needed D. CheckV removes taxonomy

> **Note**
>
> **Answer:** A. Reference databases are incomplete for viruses.

Q5. What is a good marker for viral phylogeny?

A. one homologous viral gene family B. all predicted proteins mixed together C. FASTQ quality scores D. adapter sequence

> **Note**
>
> **Answer:** A. Phylogenies require homologous sequences.

6.20 Interactive quiz: Complete pipeline

7 Visualization and Reporting

Phase 4 is like doing the lab experiment; Phase 5 is like writing the results section of the paper. A good viromics report should show **what was detected, how good the viral contigs were, how abundant they were, what they may be, what they may do, and which hosts they may infect.** In practice a report answers six questions:

1. How good were the reads?
2. How good was the assembly?
3. How many viral contigs were recovered, and how complete were they?
4. Which vOTUs were abundant?
5. What can we say about taxonomy and function?
6. What can we say about host prediction and diversity?

This chapter turns pipeline output tables into figures with R (`ggplot2`, `pheatmap`, `vegan`) and Python (`pandas`, `matplotlib`), and it pairs every figure with careful reporting language (Posit 2024).

Learning objectives — by the end of this chapter you will be able to:

- assemble the clean tables a viromics figure set is built from;
- generate the eight core result figures from example data with one script;
- read each figure and describe what it does and does not show;
- apply publication figure rules and write conservative figure legends; and
- structure an IMRaD virome report with defensible Methods and Results wording.

Every figure in this chapter is real: it was produced by the two ready-made scripts below from the small teaching tables in `data/example/`. Download either script, run it, and the exact PNGs shown here appear in `visualization/figures`. Point the same scripts at your own Phase 4 outputs to make publication figures from real data.

```
# R version (ggplot2 / pheatmap / vegan)
Rscript make_figures.R data/example visualization/figures

# Python version (pandas / matplotlib), same figures, no R needed
python make_figures.py data/example visualization/figures
```

Both scripts take two arguments: an input directory and an output directory. Swap `data/example` for your own Phase 4 folder to regenerate every figure from real results.

7.1 Expected input files

These are the Phase 4 outputs a full run consumes. Real tool paths vary, so always inspect a header before trusting a column position.

```
cd ~/viromics_course

ls checkv/checkv_samples/quality_summary.tsv          # CheckV quality
ls abundance/samples_manual_tpm.tsv                   # TPM abundance matrix
ls votus/cdhit_samples/samples_votus_95.fa            # dereplicated vOTUs
ls taxonomy/genomad_votu/                             # geNomad taxonomy
ls annotation/eggnog/samples_eggnog.emapper.annotations
ls host_prediction/iphop/                             # iPHoP host calls
```

Create a single tidy output folder so figures, tables, and scripts stay together:

```
cd ~/viromics_course
mkdir -p visualization/figures visualization/tables \
         visualization/scripts visualization/reports
```

7.2 Example teaching tables

Real tool tables are wide and messy. For teaching we derive small clean tables first; these are the exact files in `data/example/` that both scripts read. Match your figure text to these numbers.

CheckV quality summary (`checkv_quality_summary.tsv`) — eight vOTUs spanning complete genomes to short fragments; `vOTU_05` is a provirus with some flanking host genes.

contig_id	contig_length	provirus	checkv_quality	completeness
vOTU_01	48,213	No	Complete	100.0
vOTU_02	39,187	No	High-quality	94.5
vOTU_03	32,044	No	High-quality	91.2
vOTU_04	18,992	No	Medium-quality	68.3
vOTU_05	15,230	Yes	Medium-quality	61.0
vOTU_06	9,877	No	Medium-quality	52.4
vOTU_07	6,120	No	Low-quality	28.9
vOTU_08	4,003	No	Low-quality	14.2

vOTU abundance (`votu_abundance.tsv`) — TPM across two controls and two fertilized-soil samples.

votu	control_1	control_2	treatment_1	treatment_2
vOTU_01	120.4	98.7	540.2	612.9
vOTU_02	88.1	102.5	310.7	288.4

votu	control_1	control_2	treatment_1	treatment_2
vOTU_04	210.6	198.3	96.4	88.0
vOTU_05	12.3	9.8	140.5	155.2

Taxonomy (`taxonomy.tsv`) — mostly tailed phages (*Caudoviricetes*, phylum *Uroviricota*); three vOTUs stay unclassified at family level.

votu	phylum	family
vOTU_01	Uroviricota	Straboviridae
vOTU_02	Uroviricota	Autographiviridae
vOTU_03	Uroviricota	Peduoviridae
vOTU_06	Phixviricota	Microviridae
vOTU_04 / 07 / 08	unclassified	unclassified

Sample metadata (`sample_metadata.tsv`) — the grouping used for heatmap annotation and diversity.

sample	environment	treatment	nucleic_acid
control_1	soil	control	DNA
control_2	soil	control	DNA
treatment_1	soil	fertilized	DNA
treatment_2	soil	fertilized	DNA

The functional (`functional_summary.tsv`, COG categories) and host (`host_prediction.tsv`, predicted host phylum) tables round out the set and drive figures 6 and 7.

7.3 The figures

Each script starts with a shared house style and reads the tables from the input directory. Every R snippet below assumes this preamble has run.

```
suppressMessages({ library(ggplot2); library(readr); library(dplyr); library(tidyr) })

IN  <- "data/example"
OUT <- "visualization/figures"
dir.create(OUT, showWarnings = FALSE, recursive = TRUE)

# Codanics house palette
teal <- "#008b8b"; navy <- "#05043b"; terra <- "#c0432a"; gold <- "#d9a521"
quality_cols <- c("Complete"=navy, "High-quality"=teal,
                  "Medium-quality"=gold, "Low-quality"="#8a97a0")
```

```r
theme_codanics <- theme_minimal(base_size = 13) +
  theme(plot.title = element_text(face = "bold", colour = navy),
        panel.grid.minor = element_blank())
```

7.3.1 1. CheckV quality tiers

How many recovered genomes are complete versus fragmentary? This is the first thing a reviewer looks for.

```r
checkv <- read_tsv(file.path(IN, "checkv_quality_summary.tsv"), show_col_types = FALSE)
checkv$checkv_quality <- factor(
  checkv$checkv_quality,
  levels = c("Complete","High-quality","Medium-quality","Low-quality"))

g1 <- ggplot(checkv, aes(checkv_quality, fill = checkv_quality)) +
  geom_bar() +
  scale_fill_manual(values = quality_cols, guide = "none") +
  labs(title = "CheckV quality of recovered viral genomes",
       x = NULL, y = "Number of vOTUs") + theme_codanics
ggsave(file.path(OUT, "fig-checkv-quality.png"), g1, width = 7, height = 4.2, dpi = 300)
```

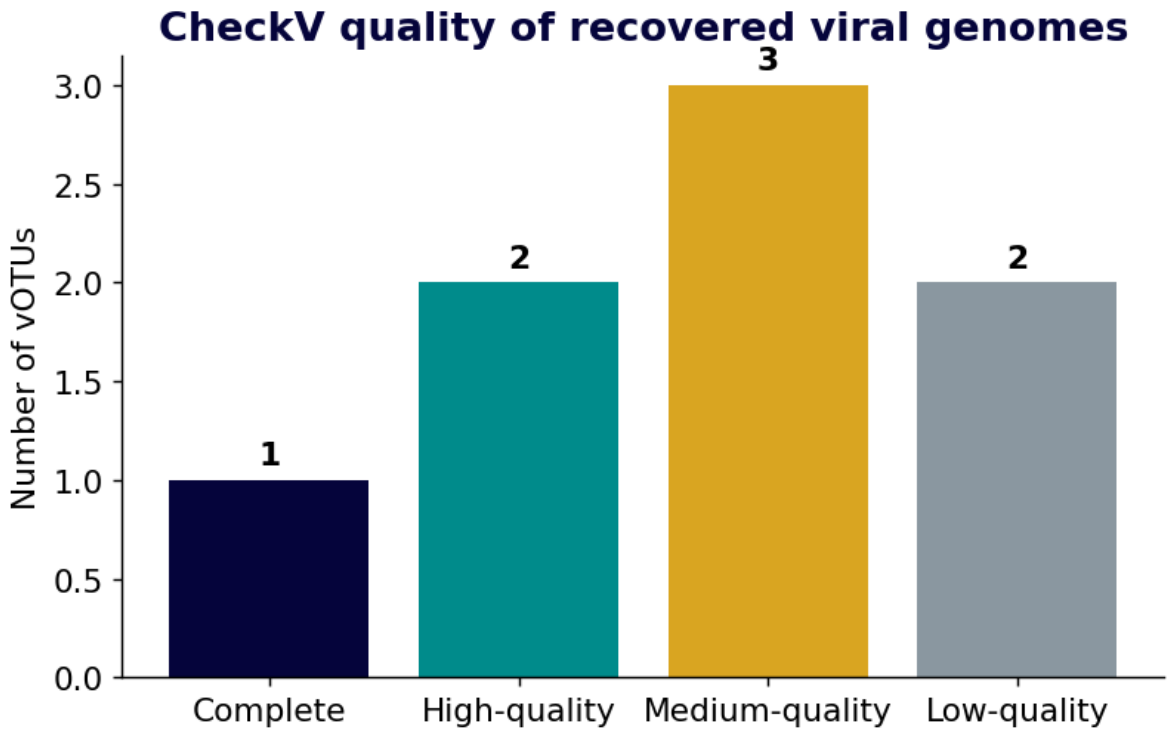


Figure 7.1: CheckV quality tiers of the eight recovered viral genomes: one Complete, two High-quality, three Medium-quality, and two Low-quality.

How to read it: the taller the left-hand bars, the more near-complete genomes you recovered; here only one genome is Complete, so most downstream claims rest on partial sequences.

7.3.2 2. Viral contig lengths

Length is a quick proxy for how much evidence each contig carries — short fragments are easy to over-interpret.

```
g2 <- ggplot(checkv, aes(reorder(contig_id, contig_length), contig_length / 1000)) +
  geom_col(fill = teal) + coord_flip() +
  labs(title = "Viral contig length", x = NULL, y = "Length (kb)") +
  theme_codanics
ggsave(file.path(OUT, "fig-contig-length.png"), g2, width = 7, height = 4.2, dpi = 300)
```

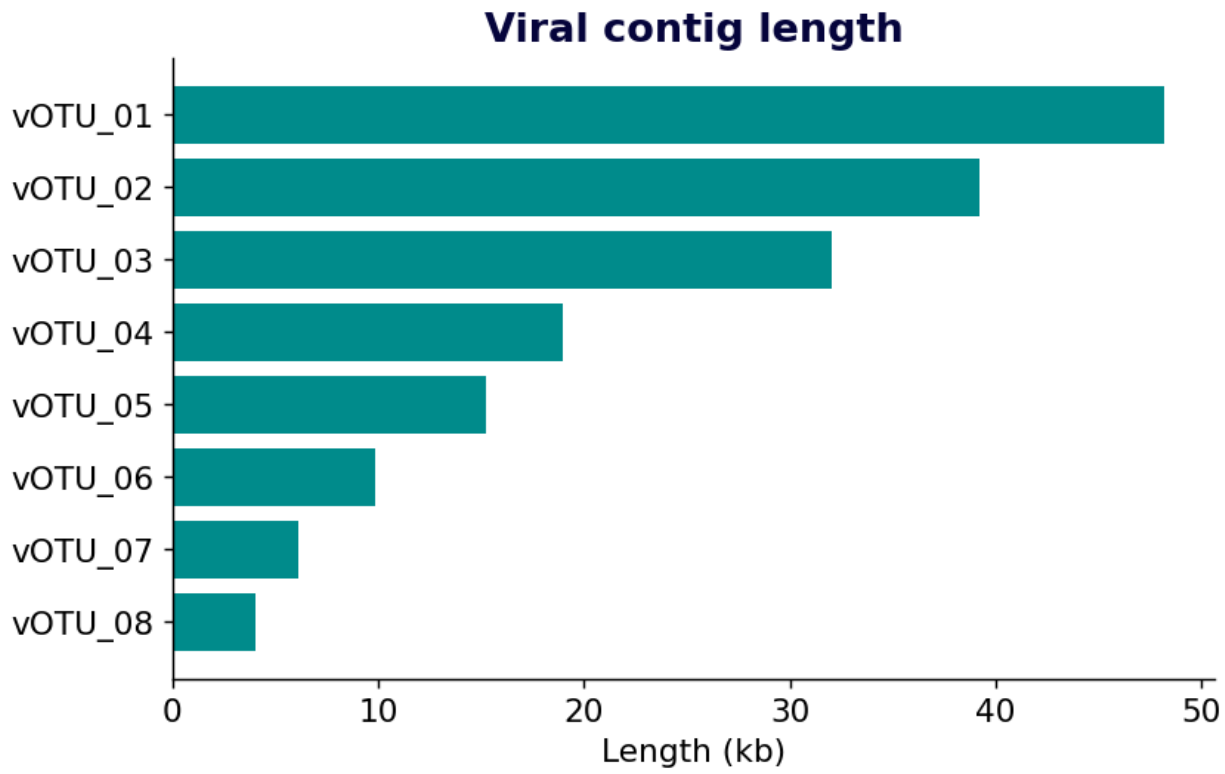


Figure 7.2: Viral contig lengths in kilobases, from vOTU_01 at 48 kb down to vOTU_08 at 4 kb.

How to read it: longer bars (top) are near-complete genomes; the short bars at the bottom coincide with the Low-quality tier and deserve cautious wording.

7.3.3 3. Mean abundance per vOTU

Which viruses dominate the community once abundance is normalized to TPM?

```
ab <- read_tsv(file.path(IN, "votu_abundance.tsv"), show_col_types = FALSE)
ab_long <- pivot_longer(ab, -votu, names_to = "sample", values_to = "tpm")
mean_ab <- ab_long %>% group_by(votu) %>% summarise(mean_tpm = mean(tpm))

g3 <- ggplot(mean_ab, aes(reorder(votu, mean_tpm), mean_tpm)) +
  geom_col(fill = navy) + coord_flip() +
  labs(title = "Mean abundance per vOTU (TPM)", x = NULL, y = "Mean TPM") +
  theme_codanics
ggsave(file.path(OUT, "fig-top-abundance.png"), g3, width = 7, height = 4.2, dpi = 300)
```

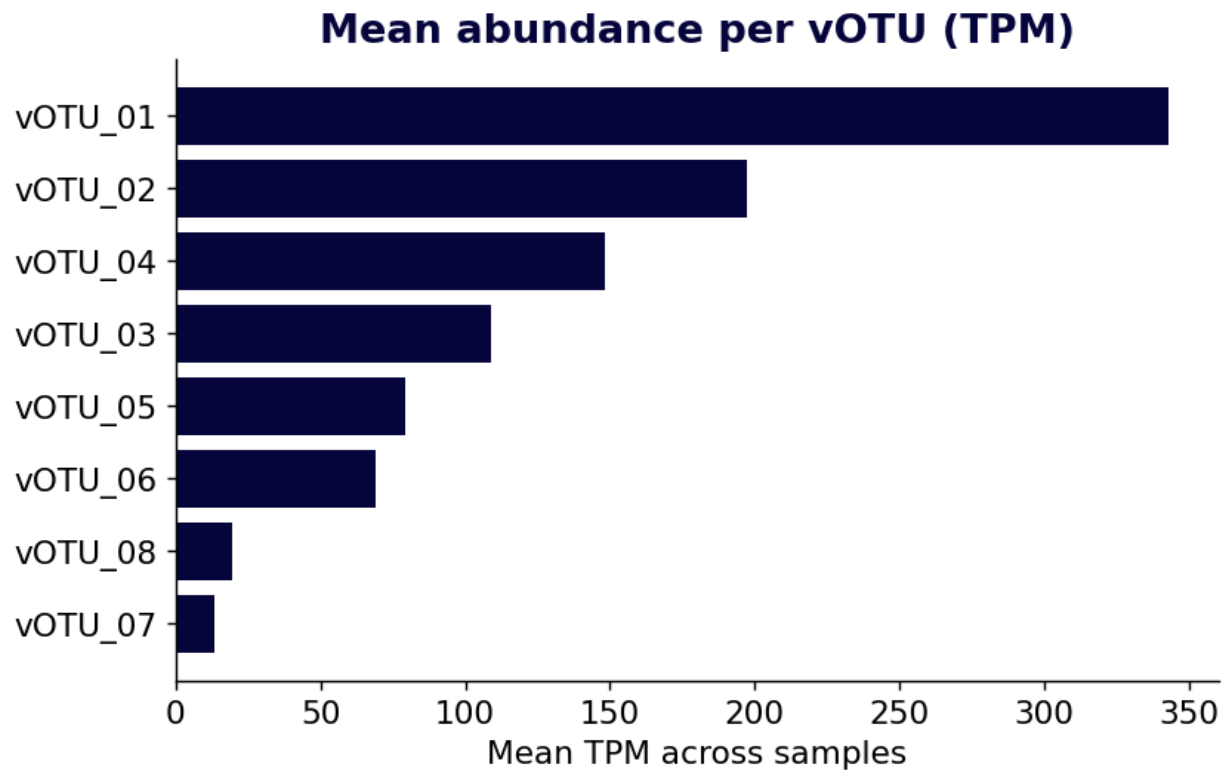


Figure 7.3: Mean TPM abundance per vOTU averaged across the four samples, with vOTU_01 the most abundant.

How to read it: the top bar (vOTU_01) is the most abundant virus on average; abundance rank does not track genome quality, so a dominant vOTU can still be only medium-quality.

7.3.4 4. Abundance heatmap with sample annotation

A heatmap shows *where* each vOTU is abundant. Values are log10(TPM + 1) so a few high-abundance viruses do not wash out the rest, and a colour bar marks the treatment groups.

```
mat <- as.matrix(ab[,-1]); rownames(mat) <- ab$votu
meta <- read_tsv(file.path(IN, "sample_metadata.tsv"), show_col_types = FALSE)
```

```r
ann <- data.frame(treatment = meta$treatment, row.names = meta$sample)

pheatmap::pheatmap(log10(mat + 1), annotation_col = ann,
                   color = colorRampPalette(c("#f7fbfb", teal, navy))(50),
                   main = "vOTU abundance (log10 TPM+1)",
                   filename = file.path(OUT, "fig-abundance-heatmap.png"),
                   width = 6.5, height = 5)
```

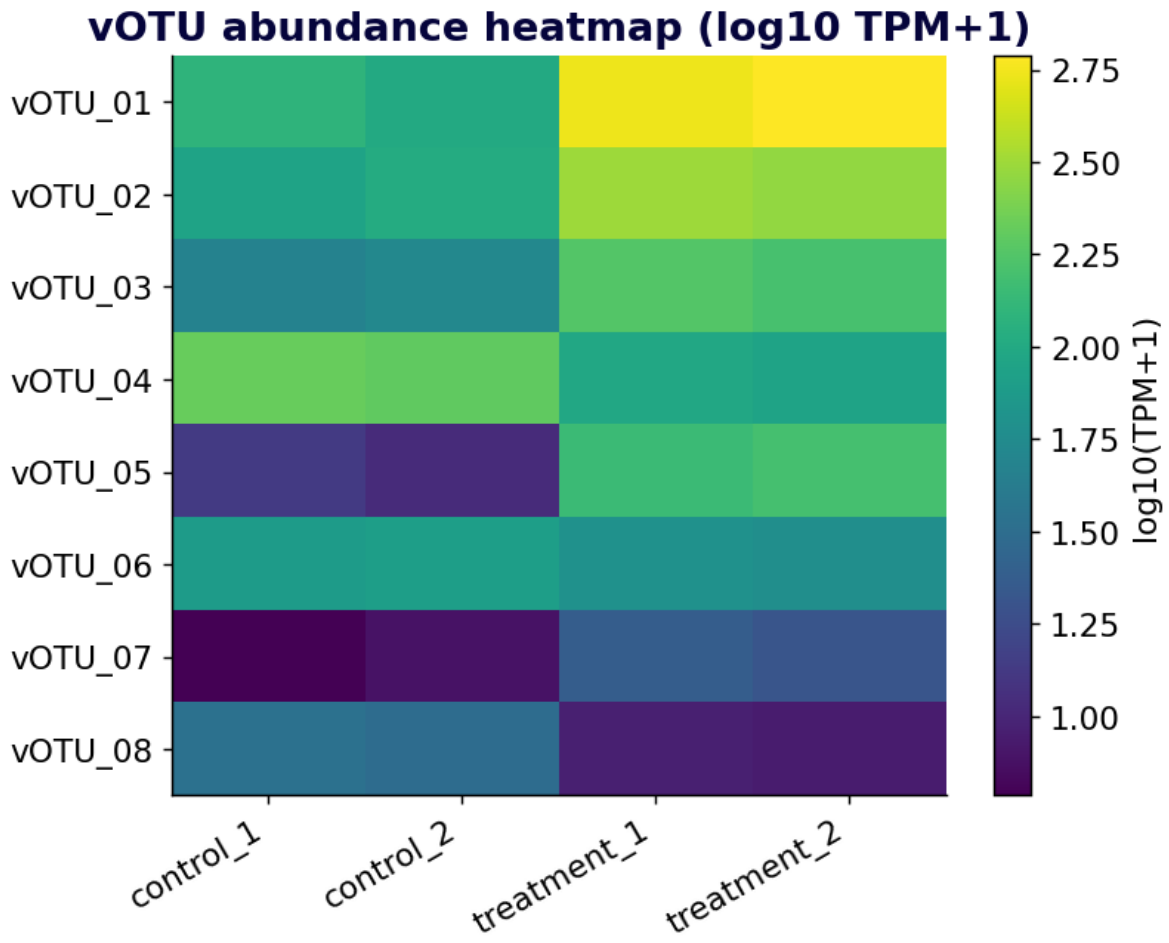


Figure 7.4: Clustered heatmap of log10(TPM + 1) vOTU abundance across four samples, annotated by treatment.

How to read it: darker cells are higher abundance; clustering that groups the two fertilized samples apart from the two controls is evidence of a treatment-associated shift, not proof of one.

7.3.5 5. Taxonomy by family

Which viral families are represented, and how much stays unclassified?

```
tax <- read_tsv(file.path(IN, "taxonomy.tsv"), show_col_types = FALSE)
g5 <- ggplot(tax, aes(forcats::fct_infreq(family))) +
  geom_bar(fill = teal) +
  labs(title = "vOTU taxonomy by family (geNomad)", x = NULL, y = "Number of vOTUs") +
  theme_codanics + theme(axis.text.x = element_text(angle = 30, hjust = 1))
ggsave(file.path(OUT, "fig-taxonomy.png"), g5, width = 7, height = 4.2, dpi = 300)
```

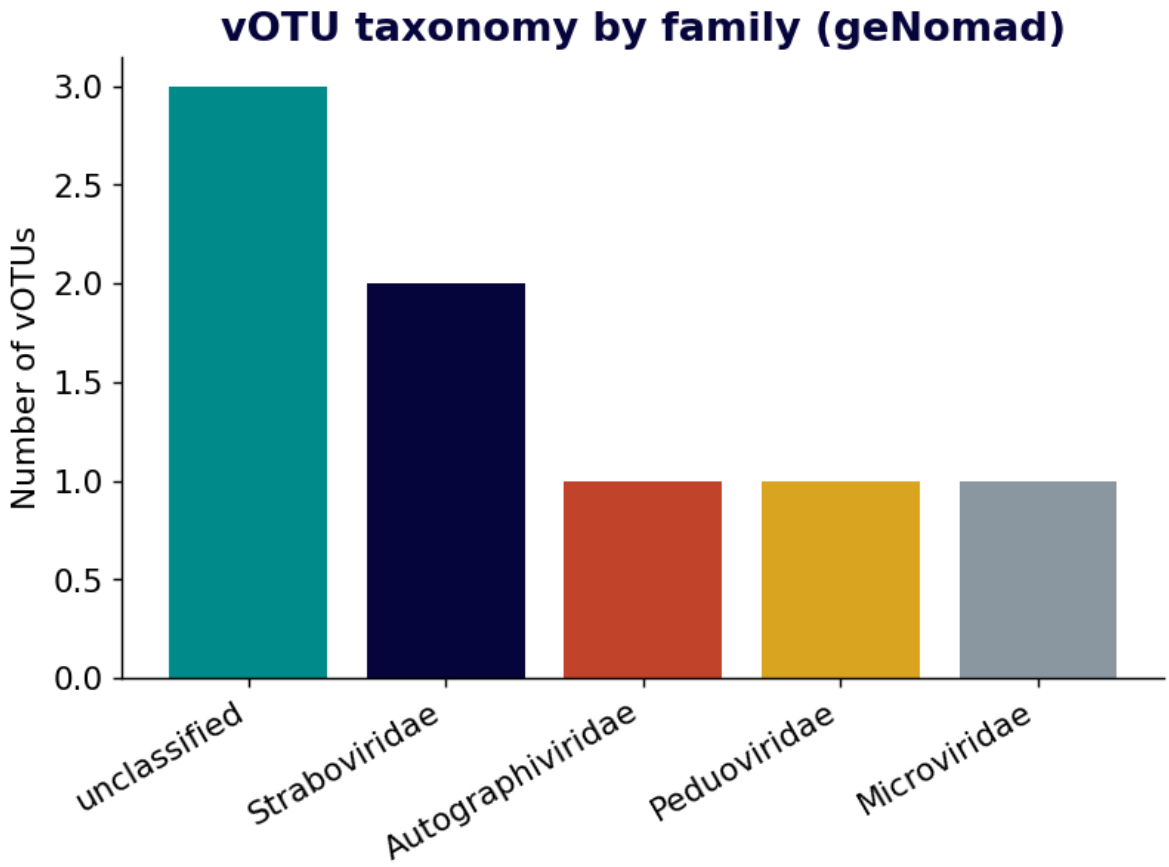


Figure 7.5: vOTU counts per viral family assigned by geNomad, with an unclassified group.

How to read it: most vOTUs are tailed phages (*Straboviridae*, *Autographiviridae*, *Peduoviridae*); a large unclassified bar is normal in viromics and should be reported, not hidden (Camargo et al. 2024).

7.3.6 6. Functional annotation by COG category

What functions do the predicted proteins fall into? eggNOG COG categories give a coarse but honest summary.

```
func <- read_tsv(file.path(IN, "functional_summary.tsv"), show_col_types = FALSE)
g6 <- ggplot(func, aes(reorder(description, count), count)) +
```

```
  geom_col(fill = teal) + coord_flip() +
  labs(title = "Functional annotation by COG category (eggNOG)",
       x = NULL, y = "Number of proteins") + theme_codanics
ggsave(file.path(OUT, "fig-function.png"), g6, width = 7, height = 4.6, dpi = 300)
```

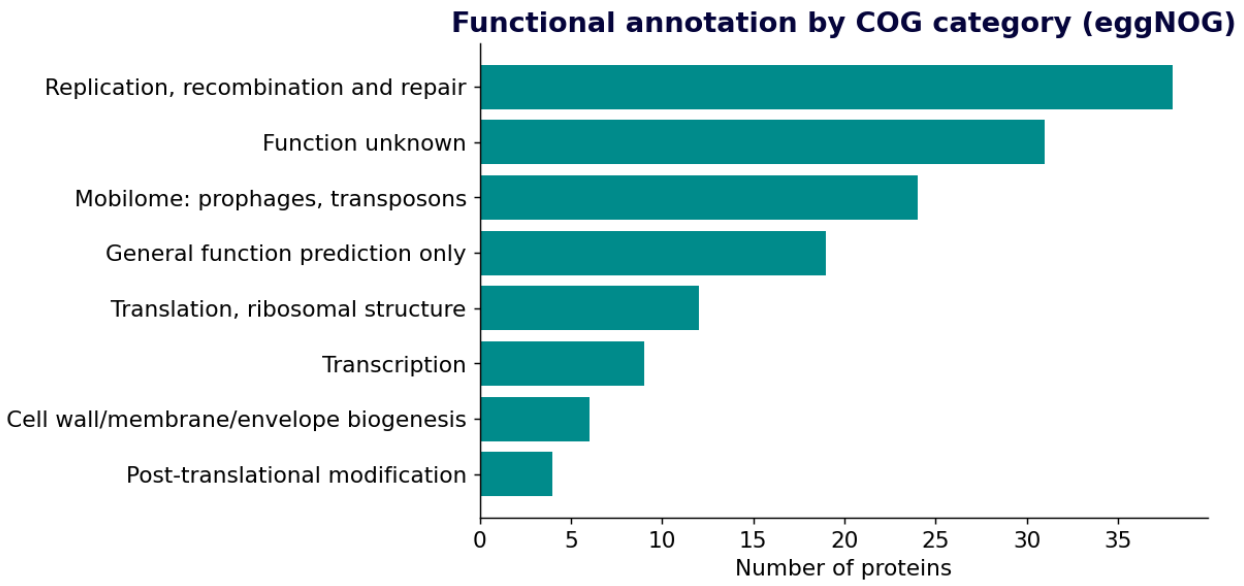


Figure 7.6: Protein counts per eggNOG COG category, led by replication/repair and a large function-unknown group.

How to read it: replication, recombination and repair (L) dominates, but "Function unknown" (S) is the second largest bar — most viral proteins have no confident functional call.

7.3.7 7. Predicted host phylum

Which host phyla do the vOTUs likely associate with? Unassigned vOTUs are dropped before counting.

```
host <- read_tsv(file.path(IN, "host_prediction.tsv"), show_col_types = FALSE)
host <- dplyr::filter(host, predicted_host_phylum != "unassigned")
g7 <- ggplot(host, aes(forcats::fct_infreq(predicted_host_phylum))) +
  geom_bar(fill = navy) +
  labs(title = "Predicted host phylum", x = NULL, y = "Number of vOTUs") +
  theme_codanics + theme(axis.text.x = element_text(angle = 20, hjust = 1))
ggsave(file.path(OUT, "fig-host.png"), g7, width = 7, height = 4.2, dpi = 300)
```

How to read it: the bars show *candidate* host phyla (Bacillota, Bacteroidota, Pseudomonadota, Actinomycetota); these are computational predictions, never confirmed infections.

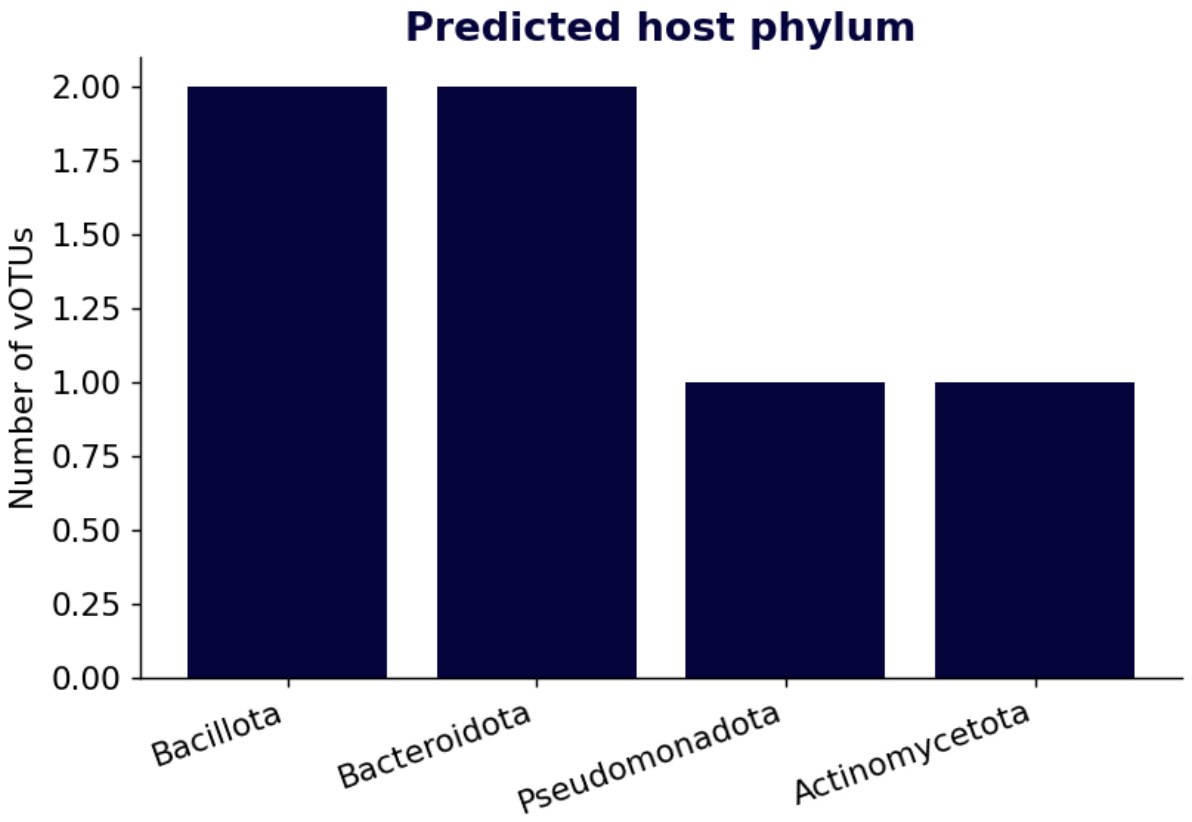


Figure 7.7: vOTU counts per predicted host phylum from iPHoP and CRISPR matching, excluding unassigned vOTUs.

7.3.8 8. Alpha diversity per sample

How rich is each sample? Observed vOTU richness is the simplest alpha-diversity metric.

```
comm <- t(as.matrix(ab[,-1])); colnames(comm) <- ab$votu
richness <- rowSums(comm > 0)
shannon  <- vegan::diversity(comm, index = "shannon")
div <- data.frame(sample = names(richness), richness, shannon)

g8 <- ggplot(div, aes(sample, richness)) + geom_col(fill = teal) +
  labs(title = "Observed vOTU richness per sample", x = NULL, y = "Observed vOTUs") +
  theme_codanics + theme(axis.text.x = element_text(angle = 20, hjust = 1))
ggsave(file.path(OUT, "fig-alpha-diversity.png"), g8, width = 6.5, height = 4.2, dpi = 300)
```

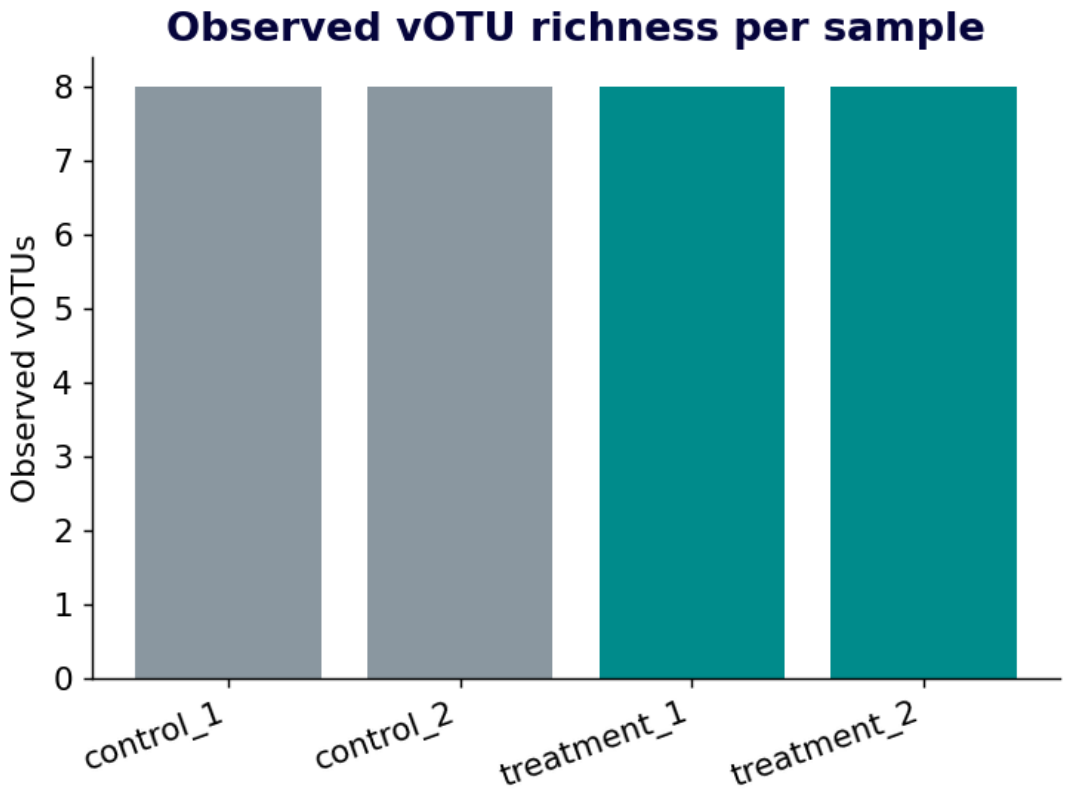


Figure 7.8: Observed vOTU richness per sample, saturated at eight in this small teaching set.

How to read it: every bar sits at eight because all vOTUs are detected in all four samples here; on real data unequal bars would flag samples with lower viral richness, and Shannon diversity (also computed above) would separate even samples with identical richness.

Estimated resources on 12 threads and 32 GB RAM: plotting takes seconds to a couple of minutes. Storage is small — usually under 500 MB for all figures and tables.

7.4 Publication figure rules

When these figures graduate from teaching to a manuscript, apply the following:

Rule	Recommendation
File type	PDF/SVG for vector graphics; PNG/TIFF at 300 dpi for raster
Font size	8–12 pt for journal figures
Labels	always label axes with units
Abundance	use TPM, RPKM, or coverage — not raw read counts alone
Heatmaps	use `log10(TPM + 1)` or row scaling
Taxonomy	do not overclassify unknown viruses
Host prediction	report as "predicted host", not confirmed host
Color	keep treatment colors consistent across every figure
Reproducibility	save the scripts and the input tables
Statistics	never show a p-value without naming the test

7.5 Figure legend template

A legend states what is plotted, how it was processed, and how to read it. Examples matched to the figures above:

```
Figure 1. CheckV quality of recovered viral genomes. Bar plot of the number of
vOTUs per CheckV category (Complete, High-quality, Medium-quality, Low-quality).

Figure 4. vOTU abundance heatmap. Heatmap of log10(TPM + 1)-transformed abundance
across samples; rows are vOTUs, columns are samples, and the top bar marks
treatment group.

Figure 7. Predicted viral host distribution. Bar plot of predicted host phyla
for vOTUs based on computational host prediction (iPHoP and CRISPR matching);
values are candidate associations, not confirmed infections.
```

7.6 Reporting language

The difference between a defensible report and an overclaim is almost always word choice. Prefer conservative phrasing, and never let software output masquerade as experimental proof.

Instead of	Write
“All contigs are confirmed viruses.”	“Putative viral contigs were identified with VirSorter2 and geNomad, then quality-assessed with CheckV.”
“The host was confirmed by software.”	“Host prediction suggested candidate associations with bacterial phyla.”
“vOTU_07 is a novel species.”	“vOTU_07 remained unclassified and represents a candidate novel viral contig.”

7.6.1 Methods wording template

```
Raw paired-end reads were quality-assessed with FastQC and summarized with
MultiQC, then trimmed with fastp. Clean reads were assembled de novo with
MEGAHIT (minimum contig length 1,000 bp) and assessed with QUAST. Viral
candidate contigs were identified with VirSorter2 and geNomad, and viral genome
quality was assessed with CheckV. Medium-quality, high-quality, and complete
contigs were dereplicated into viral operational taxonomic units (vOTUs) at 95%
nucleotide identity. Taxonomy was assigned with geNomad. Open reading frames
were predicted with Prodigal and annotated with eggNOG-mapper. Clean reads were
mapped back with Bowtie2 and TPM-normalized abundance computed with CoverM. Host
associations were predicted with iPHoP and CRISPR spacer matching. Figures were
produced in R (ggplot2, pheatmap, vegan) and Python (pandas, matplotlib).
```

7.6.2 Results wording examples

```
CheckV: The assembly yielded eight vOTUs after quality filtering. One was
complete, two high-quality, three medium-quality, and two low-quality,
indicating a mix of near-complete and fragmentary viral genomes.

Abundance: vOTU_01 and vOTU_02 were markedly more abundant in fertilized soil,
whereas vOTU_04 was enriched in the controls, suggesting a treatment-associated
shift in viral community composition.

Taxonomy: Most vOTUs were assigned to tailed bacteriophage families, while three
remained unclassified - a pattern expected given incomplete viral reference
databases.

Host prediction: Computational prediction suggested candidate associations with
Bacillota, Bacteroidota, Pseudomonadota, and Actinomycetota. These should be
read as predicted, not confirmed, host associations.
```

7.6.3 Report structure (IMRaD)

```
Title:  DNA virome analysis of paired-end metagenomic sequencing data

1. Introduction  - what viromics is, why viral communities matter, study aim
2. Methods       - sampling, sequencing, QC, assembly, viral prediction, CheckV,
                   vOTU clustering, taxonomy, annotation, abundance, host, diversity
3. Results       - read quality, assembly stats, viral recovery, CheckV quality,
                   vOTU abundance, taxonomy, function, host prediction, diversity
4. Discussion    - major patterns, novel/unclassified viruses, ecology, limitations
5. Conclusion    - main findings and future work
6. Supplementary - commands, software versions, output tables
```

Common mistakes

- **Plotting raw read counts as abundance.** Raw counts are biased by contig length and depth — use TPM, RPKM, or coverage.
- **Heatmaps without transformation.** One highly abundant vOTU can flatten all other colour; log-transform or row-scale first.
- **Overinterpreting taxonomy and host prediction.** Report unclassified contigs honestly and call host predictions candidate associations.
- **Figures with no legend or units.** A figure that cannot be traced to a table and a script is not reproducible.

7.7 Key takeaways

- A viromics report answers six questions in order — read quality, assembly quality, viral recovery and completeness, abundance, taxonomy and function, and host and diversity — and every figure should map to one of them.
- Build figures from small, tidy tables derived from Phase 4 outputs, and keep the scripts and input tables so any figure can be regenerated and traced.
- Normalize abundance (TPM, RPKM, or coverage) and log-transform heatmaps so a few dominant vOTUs do not mask the rest; never plot raw read counts as abundance.
- Report unclassified viral contigs honestly — a large "unclassified" share is normal in viromics and should be shown, not hidden (Camargo et al. 2024).
- Word choice carries the claim: describe host calls as candidate associations, quality-filter with CheckV before interpretation, and never present computational output as experimental proof (Nayfach et al. 2021).
- Follow publication figure rules — labelled axes with units, consistent treatment colours, vector formats, and legends that state what is plotted and how it was processed.

7.8 Further reading

- geNomad taxonomy and classification behaviour, including why unclassified fractions are expected (Camargo et al. 2024).
- CheckV completeness and contamination estimates that underpin defensible quality figures (Nayfach et al. 2021).
- ggplot2 documentation for building and styling the R figures in this chapter — https://ggplot2.tidyverse.org.

7.9 Chapter figure

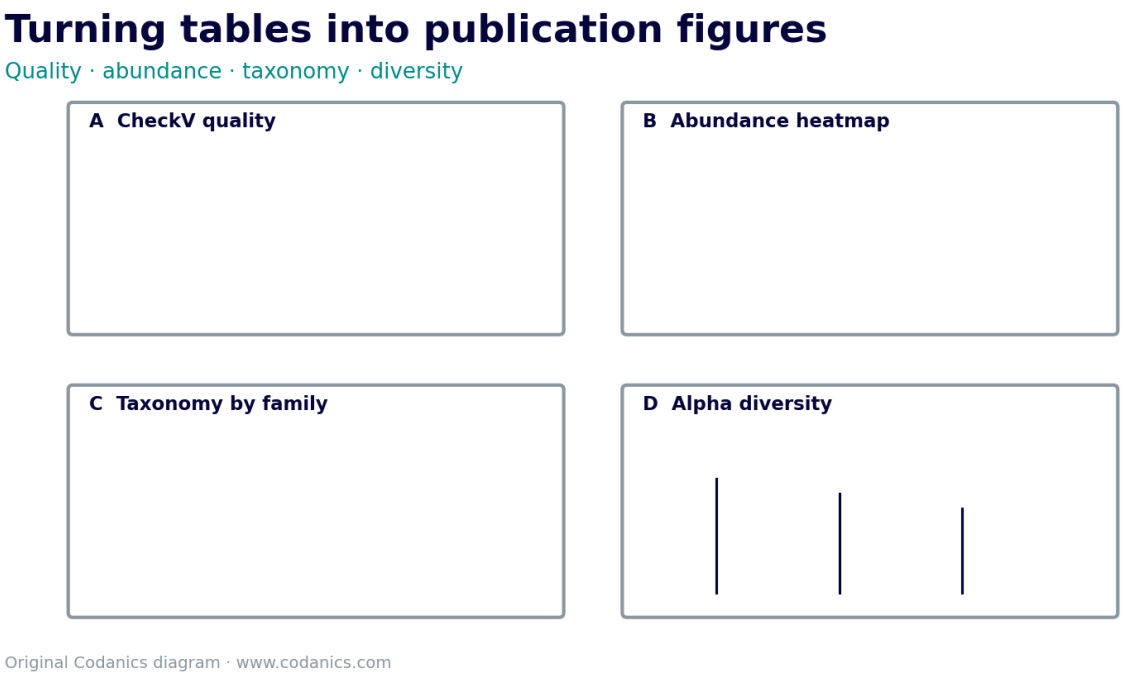


Figure 7.9: A panel of the core viromics result figures — CheckV quality, abundance heatmap, taxonomy, host prediction, and diversity — laid out as a publication figure set.

Generate this figure

Save as: `images/ch05-figure-panels.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.

Prompt: Create a clean scientific figure-panel layout arranged as a 2-by-3 grid of small viromics charts, each with a bold short title and labelled axes. Panel A: a vertical bar chart of CheckV quality tiers (Complete, High-quality, Medium-quality, Low-quality). Panel B: a horizontal bar chart of mean vOTU abundance in TPM. Panel C: a clustered heatmap of log-transformed abundance across four samples with a small treatment colour bar on top. Panel D: a bar chart of viral families including an "unclassified" bar. Panel E: a bar chart of predicted

host phyla. Panel F: a bar chart of observed vOTU richness per sample. Use teal and navy fills on a white background, thin grey axes, consistent typography, and a small "reproducible from table + script" caption ribbon along the bottom. Modern, uncluttered, journal-figure aesthetic with Codanics branding.

7.10 Quiz: Visualization and Reporting

Q1. Why use log10(TPM + 1) for heatmaps?

A. to reduce dominance of highly abundant vOTUs B. to remove all zeros only C. to convert DNA to RNA D. to calculate assembly N50

> **Note**
>
> **Answer:** A. A log transform makes abundance patterns easier to see.

Q2. Which file type is good for editable vector figures?

A. PDF or SVG B. FASTQ C. BAM D. SRA

> **Note**
>
> **Answer:** A. Vector formats are preferred for publication editing.

Q3. What should host predictions be called?

A. candidate associations B. confirmed infections C. library indexes D. assembly errors

> **Note**
>
> **Answer:** A. Computational predictions are not experimental confirmation.

Q4. What should a figure legend include?

A. what is plotted and how it was processed B. only software logo C. only file size D. no methods

> **Note**
>
> **Answer:** A. A legend should explain plotted data, transformations, and important interpretation.

Q5. Which package is used for heatmaps in this chapter?

A. pheatmap B. Bowtie2 C. CheckV D. MEGAHIT

> **Note**
>
> **Answer:** A. pheatmap creates clustered heatmaps in R.

7.11 Interactive quiz: Visualization and reporting

8 End-to-End Mini Project

This chapter is the capstone: a complete, reproducible walkthrough that takes a **real public sequencing run** from the Sequence Read Archive all the way to a compact virome report. Nothing here is simulated. You will download the same bytes any other researcher would, mine viral contigs from a mixed wastewater metagenome, judge their quality honestly, cluster them into vOTUs, measure abundance, and write up what you can — and cannot — claim.

Think of it as the difference between practicing scales and playing a whole piece. Each previous chapter drilled one technique; here they connect into a single pipeline you can run end to end.

Learning objectives — by the end of this chapter you will be able to:

- retrieve and document a public SRA run with full provenance, using `prefetch` and `fasterq-dump`;
- run a quality-control, trimming, and assembly workflow on a real paired-end metagenome;
- mine viral contigs with two complementary tools (VirSorter2 and geNomad) and combine their candidates;
- use CheckV to filter recovered sequences to medium/high/complete quality with a safe fallback;
- dereplicate contigs into vOTUs, estimate abundance, and assemble a reproducible one-page report; and
- interpret metagenomic viral mining conservatively and avoid the most common student mistakes.

> **Shotgun metagenome, not a virus-enriched virome**
>
> This run is a **shotgun wastewater metagenome**: it contains bacterial, archaeal, and eukaryotic DNA alongside viral DNA, with no particle enrichment or capsid protection step. We are *mining* viral signal from a mixed community, not measuring a purified virome. That is exactly what makes it a good teaching dataset — it forces you to separate real viral contigs from cellular background, which is the everyday reality of environmental viromics.

8.1 Run it all in one command

Once you understand each step below, the core pipeline — download through vOTU dereplication — is packaged in a single script. Download it, read it, then run it. (Annotation, abundance, figures, and the final report are then run as shown in the later steps.)

```
# After the databases and conda environments are in place:
bash run_mini_project_pipeline.sh    # or: bash ~/Downloads/run_mini_project_pipeline.sh
```

The rest of this chapter unpacks that script step by step so you understand every decision it makes.

8.2 Project title

```
Mining viral contigs from a public wastewater metagenome using Linux-based viral bioinformatics
```

8.3 Research question

```
Can we recover putative viral contigs from a public paired-end wastewater metagenome,
assess their quality, cluster them into vOTUs, estimate abundance, and generate a
compact virome report?
```

8.4 Dataset provenance

Reproducibility starts with knowing exactly what you downloaded. The Sequence Read Archive stores raw sequencing data and supports reuse of public datasets (NCBI 2026). The full provenance for the run used here is:

Field	Value
SRA run	`SRR29680455` (NCBI Sequence Read Archive 2024)
BioProject	`PRJNA527877`
Experiment	`SRX25183710`
Sample	`PHL1-P1-SS1`
Study name	PIRE: HEARD (Halting Environmental Antimicrobial Resistance Dissemination)
Sample type	wastewater secondary solids metagenome
Platform	Illumina NextSeq 500
Layout	paired-end
Read length	2 × 75 bp
Download size	~458.7 MB
Spots	7,366,184
Published	July 2, 2024

The parent project surveys metagenomes from municipal wastewater treatment plants; the selected experiment used shotgun metagenomic sequencing with Illumina NextSeq 500 2 × 75 bp paired-end reads (NCBI Sequence Read Archive 2024). `prefetch` and `fasterq-dump` are the standard SRA Toolkit commands for downloading and converting public runs (NCBI Sequence Read Archive 2022).

8.5 Expected final outputs

```
1. Raw FASTQ files from SRA          6. CheckV quality table
2. QC reports                        7. vOTU representative FASTA
3. Trimmed reads                     8. Viral abundance table
4. Metagenomic assembly              9. Basic taxonomy / annotation outputs
5. Putative viral contigs           10. Final mini virome report
```

8.6 Step 1: Create the project folder

```
mkdir -p ~/viromics_course/mini_project_wastewater
cd ~/viromics_course/mini_project_wastewater

mkdir -p raw_reads trimmed_reads qc_reports assemblies assembly_qc \
  viral_identification checkv votus taxonomy annotation abundance \
  visualization reports logs scripts databases
```

Set the environment variables the whole pipeline reuses. Defining them once, up front, keeps every later command short and consistent.

```
export PROJECT=$HOME/viromics_course/mini_project_wastewater
export THREADS=12
export SRA_ID=SRR29680455
export SAMPLE=samples

# Database locations (define before first use)
export VIROME_DB=$HOME/viromics_course/databases
export CHECKVDB=$VIROME_DB/checkv-db
export EGGNOG_DATA_DIR=$VIROME_DB/eggnog

cd $PROJECT
```

8.7 Step 2: Record the metadata

A metadata file travels with your results and makes them reproducible.

```
cat > metadata.tsv << 'EOF'
sample_id   sra_run bioproject  experiment  sample  environment sample_type platform    layout
samples SRR29680455 PRJNA527877 SRX25183710 PHL1-P1-SS1 wastewater  secondary_solids    Illumin
EOF

column -t -s $'\t' metadata.tsv
```

8.8 Step 3: Download and convert the SRA run

```
conda activate viromics-core

prefetch $SRA_ID --output-directory raw_reads

fasterq-dump \
  raw_reads/$SRA_ID \
  --split-files \
  --threads $THREADS \
  --outdir raw_reads

pigz -p $THREADS raw_reads/${SRA_ID}_1.fastq
pigz -p $THREADS raw_reads/${SRA_ID}_2.fastq

# Give the files the simple teaching names used throughout
ln -sf ${PROJECT}/raw_reads/${SRA_ID}_1.fastq.gz raw_reads/samples_R1.fastq.gz
ln -sf ${PROJECT}/raw_reads/${SRA_ID}_2.fastq.gz raw_reads/samples_R2.fastq.gz

seqkit stats raw_reads/samples_R1.fastq.gz raw_reads/samples_R2.fastq.gz
```

`seqkit` gives fast summary statistics for FASTA/FASTQ files (Shen et al. 2016).

Estimated resources on 12 threads and 32 GB RAM: download and conversion usually take 20 to 60 minutes depending on internet and disk speed. Keep 10 to 20 GB free because `fasterq-dump` writes temporary files before compression.

8.9 Step 4: QC and trimming

Inspect the raw reads first, then trim adapters and low-quality tails.

```
mkdir -p qc_reports/fastqc_raw qc_reports/multiqc_raw

fastqc -t $THREADS -o qc_reports/fastqc_raw \
  raw_reads/samples_R1.fastq.gz raw_reads/samples_R2.fastq.gz

multiqc qc_reports/fastqc_raw \
  -o qc_reports/multiqc_raw \
  -n samples_raw_multiqc.html
```

```
mkdir -p trimmed_reads qc_reports/fastp logs

fastp \
  -i raw_reads/samples_R1.fastq.gz \
  -I raw_reads/samples_R2.fastq.gz \
```

```
  -o trimmed_reads/samples_R1.fastp.fastq.gz \
  -O trimmed_reads/samples_R2.fastp.fastq.gz \
  --detect_adapter_for_pe \
  --cut_front \
  --cut_tail \
  --cut_window_size 4 \
  --cut_mean_quality 20 \
  --length_required 50 \
  --thread $THREADS \
  --html qc_reports/fastp/samples_fastp.html \
  --json qc_reports/fastp/samples_fastp.json \
  > logs/samples_fastp.log 2>&1

seqkit stats \
  raw_reads/samples_R1.fastq.gz raw_reads/samples_R2.fastq.gz \
  trimmed_reads/samples_R1.fastp.fastq.gz trimmed_reads/samples_R2.fastp.fastq.gz
```

Estimated resources on 12 threads and 32 GB RAM: 15 to 45 minutes. Storage after trimming can reach several GB.

8.10 Step 5: Assembly and assembly QC

MEGAHIT is fast and memory-light, which makes it a good first choice for a teaching assembly.

```
mkdir -p assemblies logs

megahit \
  -1 trimmed_reads/samples_R1.fastp.fastq.gz \
  -2 trimmed_reads/samples_R2.fastp.fastq.gz \
  -o assemblies/megahit_samples \
  --min-contig-len 1000 \
  -t $THREADS \
  > logs/samples_megahit.log 2>&1

export CONTIGS=$PROJECT/assemblies/megahit_samples/final.contigs.fa
seqkit stats $CONTIGS
grep -c "^>" $CONTIGS
```

```
mkdir -p assembly_qc/quast_megahit

quast.py \
  $CONTIGS \
  -o assembly_qc/quast_megahit \
  -t $THREADS \
  --min-contig 1000
```

QUAST reports contig counts, N50, and total assembly length (Gurevich et al. 2013).

Estimated resources on 12 threads and 32 GB RAM: MEGAHIT may take 30 minutes to 3 hours. Keep 30 to 80 GB free for temporary assembly files.

8.11 Step 6: Identify viral contigs with VirSorter2

No single tool catches every virus, so we use two complementary identifiers and pool their calls. Start with VirSorter2 (Guo et al. 2021).

```
conda activate virsorter2
cd $PROJECT

mkdir -p viral_identification/virsorter2 logs

virsorter run \
  -i $CONTIGS \
  -w viral_identification/virsorter2 \
  --keep-original-seq \
  --min-length 1000 \
  --min-score 0.5 \
  -j $THREADS \
  all \
  > logs/samples_virsorter2.log 2>&1

seqkit stats viral_identification/virsorter2/final-viral-combined.fa
head viral_identification/virsorter2/final-viral-score.tsv
```

8.12 Step 7: Identify viral contigs with geNomad

geNomad uses a different model and database, so it recovers candidates VirSorter2 may miss (Camargo et al. 2024).

```
conda activate genomad
cd $PROJECT

mkdir -p viral_identification/genomad logs

genomad end-to-end \
  --threads $THREADS \
  $CONTIGS \
  viral_identification/genomad \
  $VIROME_DB/genomad/genomad_db \
  > logs/samples_genomad.log 2>&1
```

```
# Locate geNomad's viral FASTA and summary outputs
find viral_identification/genomad -type f \
  | grep -Ei "virus|summary|taxonomy|fna|faa" | head -n 30
```

8.13 Step 8: Combine viral candidates

Begin with the VirSorter2 output, then append geNomad's viral FASTA if it exists.

```
conda activate viromics-core
cd $PROJECT

mkdir -p viral_identification/combined

cp viral_identification/virsorter2/final-viral-combined.fa \
   viral_identification/combined/samples_viral_candidates.fa

# If geNomad produced a virus FASTA, pool the two candidate sets
find viral_identification/genomad -name "*virus*.fna" -o -name "*virus*.fa"

cat \
  viral_identification/virsorter2/final-viral-combined.fa \
  $(find viral_identification/genomad -name "*virus*.fna" | head -n 1) \
  > viral_identification/combined/samples_viral_candidates_vs2_genomad.fa

seqkit stats viral_identification/combined/*.fa
```

Point the pipeline at the combined candidate set (fall back to VirSorter2 only if geNomad produced nothing):

```
export VIRAL_CANDIDATES=$PROJECT/viral_identification/combined/samples_viral_candidates_vs2_ger
[ -s "$VIRAL_CANDIDATES" ] || export VIRAL_CANDIDATES=$PROJECT/viral_identification/combined/sa
echo "Using candidates: $VIRAL_CANDIDATES"
```

8.14 Step 9: Assess viral quality with CheckV

CheckV estimates completeness and contamination and trims host flanks from proviruses (Nayfach et al. 2021). This is the single most important quality gate in the whole pipeline.

```
conda activate checkv
cd $PROJECT

mkdir -p checkv/checkv_samples logs

checkv end_to_end \
```

```
  $VIRAL_CANDIDATES \
  checkv/checkv_samples \
  -t $THREADS \
  -d $CHECKVDB \
  > logs/samples_checkv.log 2>&1

# Combine CheckV virus and provirus sequences
cat \
  checkv/checkv_samples/viruses.fna \
  checkv/checkv_samples/proviruses.fna \
  > checkv/checkv_samples/samples_checkv_combined_viral.fna

seqkit stats checkv/checkv_samples/samples_checkv_combined_viral.fna
```

Estimated resources on 12 threads and 32 GB RAM: viral identification plus CheckV can take 30 minutes to 4 hours. Storage need is usually under 20 GB beyond the databases.

8.15 Step 10: Filter to medium/high/complete quality

Never trust column positions from memory — CheckV's layout can change between versions. Always inspect the header first.

```
head -n 1 checkv/checkv_samples/quality_summary.tsv | tr '\t' '\n' | nl
```

In current CheckV, the quality label lives in column 8. Extract the IDs of contigs judged Medium-quality or better:

```
awk -F'\t' '
NR==1 {next}
$8=="Medium-quality" || $8=="High-quality" || $8=="Complete" {
  print $1
}' checkv/checkv_samples/quality_summary.tsv \
> checkv/checkv_samples/samples_medium_high_complete_ids.txt

seqkit grep \
  -f checkv/checkv_samples/samples_medium_high_complete_ids.txt \
  checkv/checkv_samples/samples_checkv_combined_viral.fna \
  > checkv/checkv_samples/samples_medium_high_complete_viral.fna

seqkit stats checkv/checkv_samples/samples_medium_high_complete_viral.fna
```

A shotgun metagenome may recover **no** medium-or-better viral genomes, which would leave the filtered file empty and break every downstream step. Guard against that with a simple fallback — if the filtered file is non-empty use it, otherwise keep all CheckV viral contigs:

```
if [ -s checkv/checkv_samples/samples_medium_high_complete_viral.fna ]; then
  export VIRAL_FASTA=$PROJECT/checkv/checkv_samples/samples_medium_high_complete_viral.fna
else
  export VIRAL_FASTA=$PROJECT/checkv/checkv_samples/samples_checkv_combined_viral.fna
fi
echo "Using viral FASTA: $VIRAL_FASTA"
```

8.16 Step 11: Dereplicate into vOTUs

Cluster near-identical contigs into **viral operational taxonomic units (vOTUs)** at 95% nucleotide identity over 85% of the shorter sequence — the community-standard (MIUViG) species-level threshold (Roux et al. 2019).

```
conda activate viromics-core
cd $PROJECT

mkdir -p votus/cdhit_samples

cd-hit-est \
  -i $VIRAL_FASTA \
  -o votus/cdhit_samples/samples_votus_95.fa \
  -c 0.95 \
  -aS 0.85 \
  -G 0 \
  -g 1 \
  -T $THREADS \
  -M 0

export VOTU_FASTA=$PROJECT/votus/cdhit_samples/samples_votus_95.fa
seqkit stats $VOTU_FASTA
```

Estimated resources on 12 threads and 32 GB RAM: usually minutes for this mini project. Storage under 5 GB.

8.17 Step 12: Gene prediction and functional annotation

Predict genes on the vOTU representatives with Prodigal (Hyatt et al. 2010), then annotate with eggNOG-mapper.

```
mkdir -p annotation/prodigal

prodigal \
  -i $VOTU_FASTA \
  -a annotation/prodigal/samples_votus.faa \
```

```
  -d annotation/prodigal/samples_votus.genes.fna \
  -o annotation/prodigal/samples_votus.gff \
  -f gff \
  -p meta

seqkit stats annotation/prodigal/samples_votus.faa
```

```
conda activate eggnog
cd $PROJECT

mkdir -p annotation/eggnog logs

emapper.py \
  -i annotation/prodigal/samples_votus.faa \
  --itype proteins \
  -m diamond \
  --cpu $THREADS \
  --data_dir $EGGNOG_DATA_DIR \
  -o samples_eggnog \
  --output_dir annotation/eggnog \
  > logs/samples_eggnog.log 2>&1

grep -v "^#" annotation/eggnog/samples_eggnog.emapper.annotations | head
```

Estimated resources on 12 threads and 32 GB RAM: Prodigal takes seconds to minutes. eggNOG can take minutes to hours depending on protein count and database storage speed.

8.18 Step 13: Abundance mapping

Map trimmed reads back to the vOTUs with Bowtie2 (Langmead and Salzberg 2012), sort with samtools (Danecek et al. 2021), then compute coverage and TPM with CoverM.

```
conda activate viromics-core
cd $PROJECT

mkdir -p abundance/bowtie2_index abundance/logs

bowtie2-build $VOTU_FASTA abundance/bowtie2_index/samples_votus

bowtie2 \
  -x abundance/bowtie2_index/samples_votus \
  -1 trimmed_reads/samples_R1.fastp.fastq.gz \
  -2 trimmed_reads/samples_R2.fastp.fastq.gz \
  -p $THREADS \
  --very-sensitive \
```

```
  2> abundance/logs/samples_bowtie2_mapping.log \
| samtools view -bS - \
| samtools sort -@ $THREADS -o abundance/samples_vs_votus.sorted.bam

samtools index abundance/samples_vs_votus.sorted.bam
samtools flagstat abundance/samples_vs_votus.sorted.bam > abundance/samples_flagstat.txt

coverm contig \
  --bam-files abundance/samples_vs_votus.sorted.bam \
  --methods mean covered_fraction count tpm \
  --threads $THREADS \
  > abundance/samples_coverm_contig.tsv
```

Estimated resources on 12 threads and 32 GB RAM: 15 minutes to 2 hours. BAM files can require several GB.

8.19 Step 14: Report tables

Again, inspect the CheckV header before slicing columns, then build clean, minimal tables for reporting.

```
mkdir -p reports/tables visualization/figures

head -n 1 checkv/checkv_samples/quality_summary.tsv | tr '\t' '\n' | nl

awk -F'\t' '
BEGIN{OFS="\t"}
NR==1 {print "contig_id","length","completeness","quality"}
NR>1  {print $1,$2,$10,$8}
' checkv/checkv_samples/quality_summary.tsv \
> reports/tables/checkv_quality_clean.tsv

seqkit fx2tab -n -l $VOTU_FASTA > reports/tables/votu_lengths.tsv
cp abundance/samples_coverm_contig.tsv reports/tables/votu_abundance_coverm.tsv
```

8.20 Step 15: Figures

The two figures below are **illustrative example outputs**, generated by `scripts/make_figures.py` from the teaching tables in `data/example/`. On your own run, the exact bars and lengths will depend on how much viral signal this dataset yields on your machine. The short R snippet that follows shows how to build the same two figures from *your* Step 14 report tables.

This mini-project snippet reads *your* Step 14 report tables and writes both figures into `visualization/figures`:

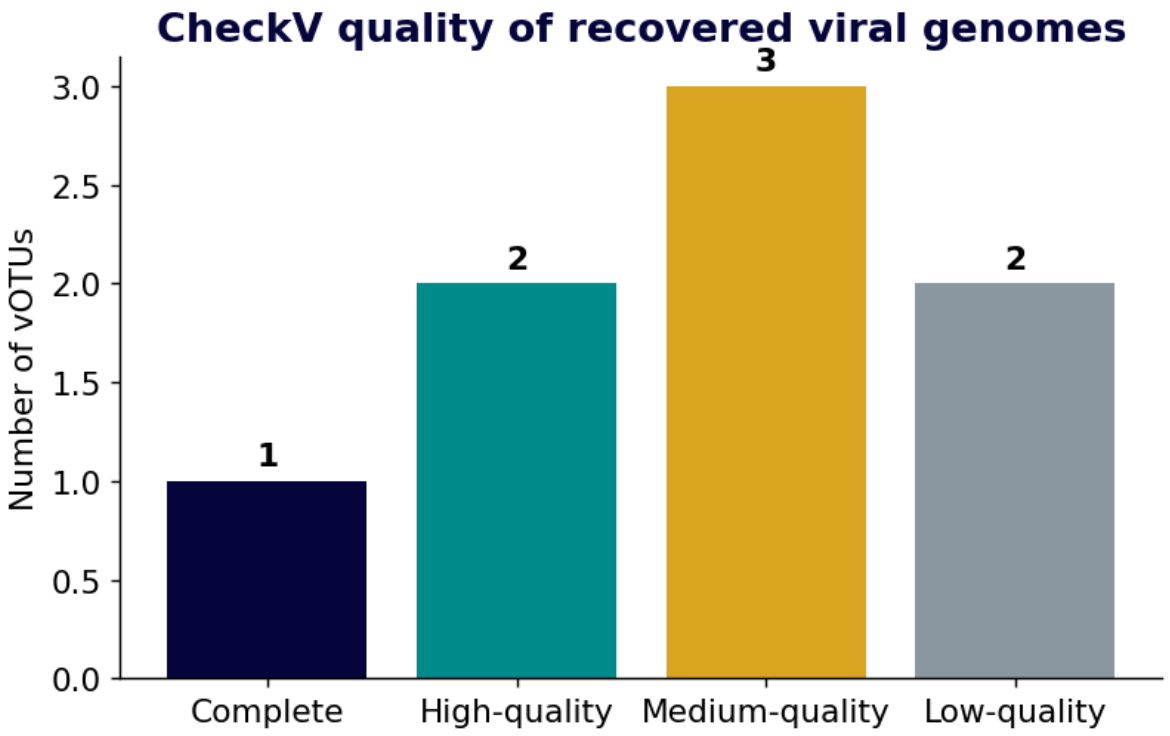


Figure 8.1: CheckV quality of the recovered vOTUs

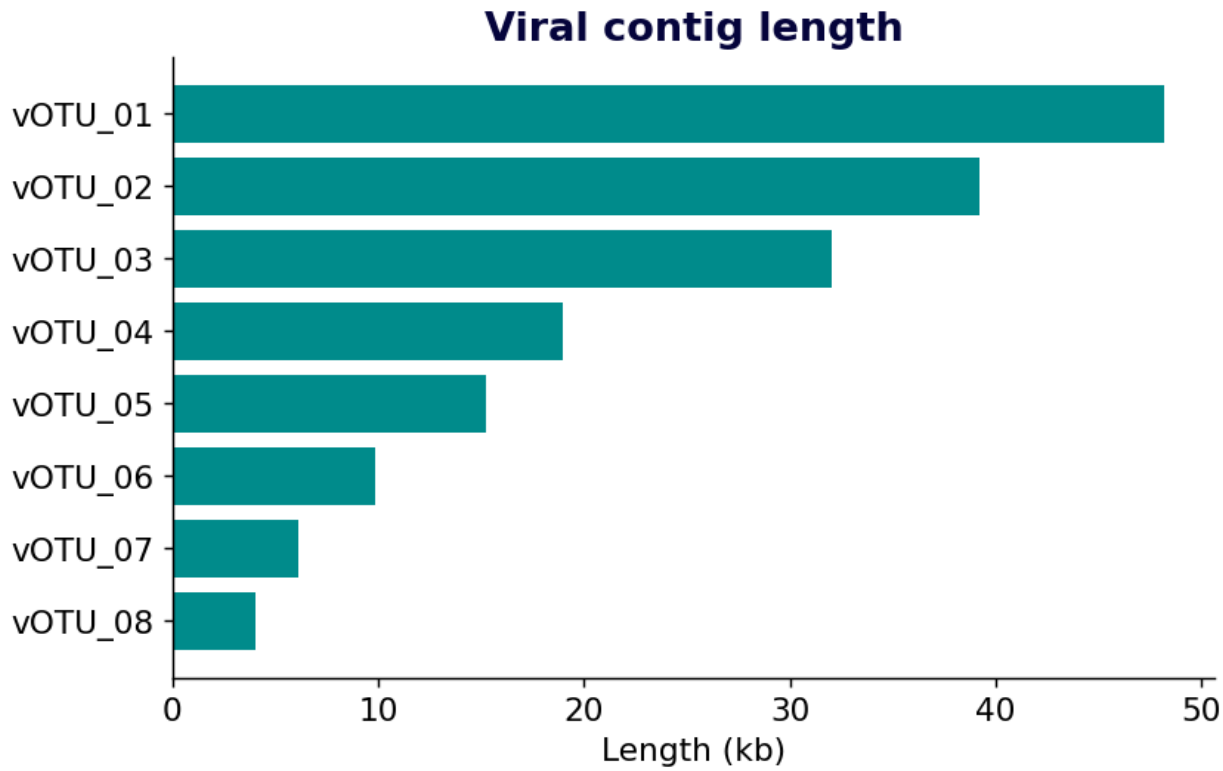


Figure 8.2: Viral contig lengths

```r
library(readr); library(dplyr); library(ggplot2)

# Figure 1: CheckV quality categories
checkv <- read_tsv("reports/tables/checkv_quality_clean.tsv", show_col_types = FALSE)
ggplot(count(checkv, quality), aes(quality, n)) +
  geom_col(width = 0.7) +
  labs(title = "CheckV quality of recovered vOTUs",
       x = "Quality category", y = "Number of viral contigs") +
  theme_bw(base_size = 12)
ggsave("visualization/figures/fig-checkv-quality.png", width = 6, height = 4, dpi = 300)

# Figure 2: viral contig length distribution
lengths <- read_tsv("reports/tables/votu_lengths.tsv",
                    col_names = c("votu_id", "length"), show_col_types = FALSE)
ggplot(lengths, aes(length / 1000)) +
  geom_histogram(bins = 20) +
  labs(title = "Viral contig lengths", x = "Length (kb)", y = "Number of vOTUs") +
  theme_bw(base_size = 12)
ggsave("visualization/figures/fig-contig-length.png", width = 6, height = 4, dpi = 300)
```

```
# Regenerate the illustrative example figures shown above
python scripts/make_figures.py

# ...or save the snippet above as a script and run it on your own tables
ls -lh visualization/figures/
```

8.21 Step 16: Final report

The reporting script gathers the key statistics and file paths into a single plain-text summary you can hand in with your deliverables.

```
bash scripts/create_report_summary.sh "$PROJECT" "$SAMPLE"
cat reports/${SAMPLE}_viromics_summary.txt
```

The script records raw, trimmed, assembly, viral, and vOTU statistics; the paths of every important output (MultiQC, fastp, QUAST, VirSorter2, geNomad, CheckV, vOTU FASTA, abundance table, figures); and a short interpretation guide reminding the reader that this is a shotgun metagenome and that predictions are putative.

8.22 Teaching interpretation for students

Use these points when explaining — and grading — the project:

1. **This is metagenomic viral mining, not a pure virome experiment.** The library contains bacteria, archaea, eukaryotic DNA, and viral DNA all mixed together. Every viral contig has to be pulled out of that background, so "how much virus did we find" is really "how much viral signal survived identification and quality filtering."
2. **VirSorter2 and geNomad predict candidates; they do not confirm viruses.** Each is a probabilistic classifier. Combining them raises sensitivity, but a call from either tool is still a hypothesis.
3. **CheckV is essential.** Many recovered contigs are incomplete fragments or carry host flanks. Reporting a fragment as if it were a finished genome is the fastest way to overstate a result. Completeness and contamination estimates are what make the recovered set defensible.
4. **A vOTU is a clustered viral unit, not a named species.** Dereplicating at ~95% nucleotide identity gives an operational, species-level grouping. It says "these contigs represent the same viral population," not "this is *Escherichia* phage X."
5. **Coverage and TPM measure abundance; assembly only measures recovery.** A contig existing tells you it assembled well; its TPM tells you how common those reads were. Do not confuse "we assembled it" with "it was abundant."
6. **Taxonomy and host prediction must stay conservative — especially for wastewater.** Wastewater pools human, animal, plant, and environmental sources, so a computational host or taxonomy call is a candidate association, not a confirmed one. Prefer family-level (or higher) labels for novel contigs.
7. **Report the protocol, not just the numbers.** Because a virome is protocol-dependent, your write-up should state that this is a shotgun (non-enriched) metagenome so readers can calibrate what "we recovered N vOTUs" actually means.

⚠ Common mistakes

- **Skipping the CheckV filter** and reporting every VirSorter2/geNomad hit as a "virus."
- **Hard-coding CheckV column numbers** instead of checking the header with `head -n1 ... | tr '\t' '\n' | nl` — the layout shifts between versions.
- **Letting an empty medium/high/complete file crash the pipeline** because you forgot the fallback `export VIRAL_FASTA` branch.
- **Claiming confirmed hosts or species** from computational predictions on a mixed wastewater sample.
- **Confusing assembly recovery with abundance** — a long contig is not necessarily a common one.
- **Reporting raw read counts as "the virome"** without noting this is a non-enriched shotgun metagenome.

8.23 Student deliverables

Ask students to submit:

```
1. raw MultiQC report
2. trimmed fastp HTML report
3. QUAST assembly report
4. VirSorter2 viral score table (and geNomad summary if produced)
```

```
5. CheckV quality_summary.tsv
6. vOTU FASTA file
7. CoverM abundance table
8. two figures:
   - CheckV quality bar plot
   - viral contig length distribution
9. one-page interpretation report
```

Suggested report title:

```
Viral contig recovery and quality assessment from a public wastewater metagenome
```

8.24 Key takeaways

- Reproducibility starts with provenance: document the exact SRA run (here `SRR29680455`, BioProject PRJNA527877) and every tool and parameter before reporting a single result (NCBI Sequence Read Archive 2024).
- This is shotgun viral mining, not a purified virome — viral contigs must be separated from bacterial, archaeal, and eukaryotic background, so "how much virus we found" means "how much viral signal survived identification and quality filtering".
- Two complementary identifiers raise sensitivity: VirSorter2 and geNomad use different models, and their pooled candidates outperform either alone (Guo et al. 2021; Camargo et al. 2024).
- CheckV is the key quality gate — completeness and contamination estimates decide which contigs are defensible, and a fallback branch prevents an empty medium/high/complete file from crashing downstream steps (Nayfach et al. 2021).
- A vOTU is a clustered, species-level viral population (~95% ANI), not a named species; coverage and TPM measure abundance, while assembly only measures recovery.
- Keep taxonomy and host predictions conservative, especially for pooled wastewater sources, and always inspect a table header before slicing columns by position.

8.25 Further reading

- The SRA run record and its metadata for the wastewater dataset used throughout this chapter (NCBI Sequence Read Archive 2024).
- Overview of the Sequence Read Archive and how public runs are retrieved and reused (NCBI 2026).
- CheckV for viral genome quality assessment (Nayfach et al. 2021) and MEGAHIT for the metagenomic assembly step (Li et al. 2015).
- NCBI SRA Toolkit documentation for `prefetch` and `fasterq-dump` — https://github.com/ncbi/sra-tools.

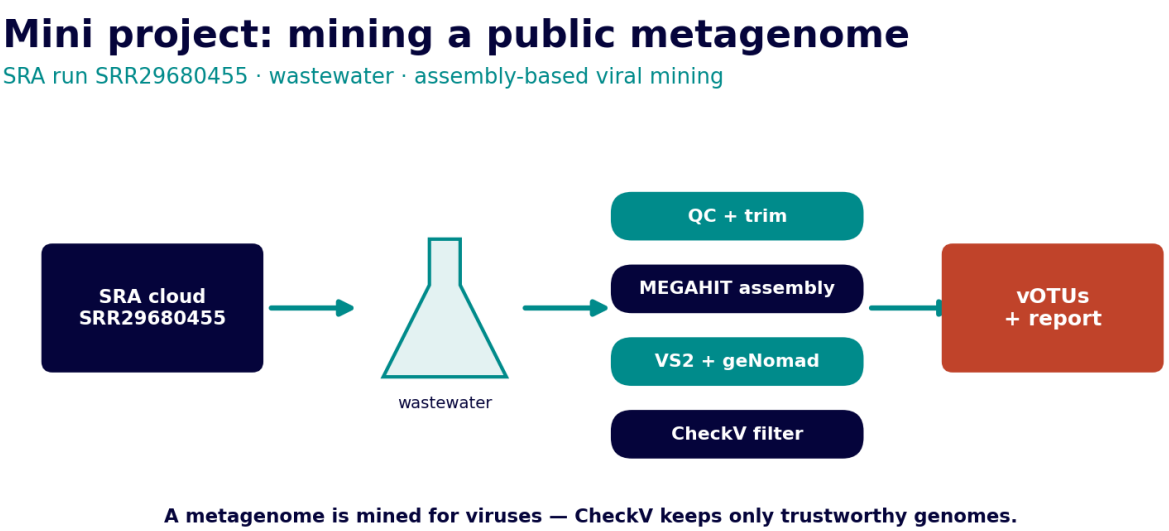


Figure 8.3: An end-to-end wastewater viral-mining pipeline, from a public SRA run through QC, assembly, dual viral identification, CheckV, vOTU clustering, and abundance to a final report.

8.26 Chapter figure

Generate this figure

Save as: `images/ch06-wastewater-miniproject.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean educational workflow infographic for a wastewater viral-mining mini project, laid out left to right as a horizontal pipeline of connected labeled stages. Start with a small wastewater-treatment icon and a database cylinder labeled "Public SRA run SRR29680455", then arrows through boxes labeled in order: "Download (prefetch / fasterq-dump)", "QC and trimming (FastQC, fastp)", "Assembly (MEGAHIT)", "Viral identification (VirSorter2 + geNomad)" shown as two parallel boxes merging into "Combined candidates", "Quality control (CheckV)", "vOTU clustering (95% ANI, CD-HIT-EST)", "Abundance (Bowtie2 + CoverM)", and ending in a document icon labeled "Virome report + figures". Use teal and navy Codanics branding, rounded boxes, clear arrows, and small monochrome icons for each stage. No photorealism, white background, readable sans-serif labels.

8.27 Quiz: Mini Project

Q1. What is the SRA run used in this mini project?

A. SRR29680455 B. SRR000001 C. ERR123456 D. SAMN00000

> **Note**
>
> **Answer:** A. The selected public run is SRR29680455 (BioProject PRJNA527877, experiment SRX25183710, sample PHL1-P1-SS1).

Q2. Why is this dataset described as viral mining?

A. it is a shotgun metagenome, not virus-enriched B. it contains only pure viral particles C. it has no bacterial reads D. it is a PCR assay

> **Note**
>
> **Answer:** A. Viral contigs are mined from mixed metagenomic reads that also contain cellular DNA.

Q3. Which command converts SRA data to FASTQ?

A. fasterq-dump B. fastqc C. quast.py D. iqtree

> **Note**
>
> **Answer:** A. `fasterq-dump` converts prefetched SRA data to FASTQ.

Q4. Why does the pipeline run both VirSorter2 and geNomad?

A. they use different models and recover complementary candidates B. one converts FASTQ and the other assembles C. geNomad replaces CheckV D. only to double the runtime

> **Note**
>
> **Answer:** A. No single identifier catches every virus, so their candidates are combined before quality control.

Q5. What should the CheckV step guard against when filtering to medium/high/complete?

A. an empty filtered file breaking downstream steps B. too many complete genomes C. missing raw reads D. adapter contamination

> **Note**
>
> **Answer:** A. A shotgun metagenome may yield no medium-or-better genomes, so a fallback `export VIRAL_FASTA` keeps the pipeline running on all CheckV viral contigs.

8.28 Interactive quiz: Mini project

9 Practice Questions and Chapter Quizzes

This chapter collects learner-focused practice that mirrors the full viromics workflow, from Linux basics to viral abundance. Work through one topic at a time, ideally right after finishing the matching chapter. Each exercise is short and self-contained, and every quiz answer is hidden behind a collapsible box so you can test yourself before checking.

Learning objectives — by the end of this chapter you will be able to:

- reproduce small command-line tasks that reinforce each stage of the pipeline;
- filter, count, and summarize tab-separated data with `awk`, `cut`, `sort`, and `seqkit`;
- recall which tool solves which problem across QC, assembly, viral identification, quality control, taxonomy, and abundance; and
- self-assess your progress with both static and interactive quizzes.

> **How to use this chapter**
>
> Some exercises read files produced by the main pipeline (assemblies, abundance tables, BAMs). Run them **only if you have run the pipeline**; otherwise substitute the ready-made tables in `data/example/`, which have the same columns and let every command run without a full analysis.

9.1 Fundamentals practice

9.1.1 Exercise

Create a TSV file with five viral genome types and use `awk` to print only RNA viruses.

```
cat > genomes.tsv << 'EOF'
virus   genome
T4  dsDNA
Influenza   -ssRNA
Coronavirus +ssRNA
Rotavirus   dsRNA
Microvirus  ssDNA
EOF

awk -F'\t' 'NR==1 || $2 ~ /RNA/' genomes.tsv
```

9.1.2 Quiz

Q1. What is the best one-line definition of a virome?

A. all bacteria in a sample B. the viral community associated with a sample or environment C. only the viruses that cause disease D. only RNA viruses

> **Note**
>
> **Answer:** B. A virome is the viral community associated with a sample or environment. In practice it is an operational measurement — you detect only the protocol-dependent subset of viruses your workflow can capture.

Q2. Viromics has no equivalent of the bacterial 16S rRNA gene. What does this imply for viral identification?

A. viruses cannot be sequenced at all B. a single universal marker gene can classify every virus C. identification must combine several imperfect signals such as hallmark genes, genome structure, and reference similarity D. only circular genomes can be identified

> **Note**
>
> **Answer:** C. There is no universal viral barcode, so tools rely on multiple lines of evidence together: viral hallmark genes, gene density and orientation, sequence composition, genome architecture, coverage, and host-association signals.

Q3. In one sentence, what is a prophage, and what sequence signal hints at one inside a bacterial contig?

> **Note**
>
> **Answer:** A prophage is the genome of a temperate phage persisting in or alongside a host genome, usually integrated into the host chromosome. On a contig it appears as a viral region flanked by host DNA, often with an integrase or recombinase gene and attachment (*att*) sites — which is why tools like CheckV trim host flanks from proviruses.

9.2 Experimental design practice

9.2.1 Exercise

Make a metadata table with two groups, three replicates per group, and one extraction blank, then count samples per control type.

```
cat > metadata.tsv << 'EOF'
sample_id   group   replicate   control_type
A1  A   1   biological_sample
A2  A   2   biological_sample
A3  A   3   biological_sample
```

```
B1  B   1   biological_sample
B2  B   2   biological_sample
B3  B   3   biological_sample
Blank1  none    NA  extraction_blank
EOF

awk -F'\t' 'NR>1 {count[$4]++} END {for (x in count) print x, count[x]}' metadata.tsv
```

The `awk` command tallies the fourth column and prints each control type with its count. Including an extraction blank lets you flag reagent and laboratory contaminants during analysis.

9.3 Linux setup practice

9.3.1 Exercise

Check whether the core conda environment exists before starting a run.

```
conda env list | grep viromics-core || echo "Environment missing"
```

If the environment is missing, create it before continuing. The same pattern (`grep <name> || echo missing`) works for `virsorter2`, `genomad`, `checkv`, and the other environments used across the book.

9.4 Pipeline practice

9.4.1 Exercise

Count assembled contigs longer than 5 kb. Run this **if you have run the pipeline**; otherwise use the example assembly in `data/example/` (for instance `data/example/final.contigs.fa`).

```
seqkit seq -m 5000 assemblies/megahit_samples/final.contigs.fa \
  > assemblies/megahit_samples/contigs_gt5kb.fa

grep -c "^>" assemblies/megahit_samples/contigs_gt5kb.fa
```

`seqkit seq -m 5000` keeps sequences of at least 5000 bp, and `grep -c "^>"` counts FASTA headers. Length filtering removes many short, low-evidence contigs before viral identification.

9.5 Visualization practice

9.5.1 Exercise

Print the top three vOTUs by TPM using Python. Use your own abundance table **if you have run the pipeline**; otherwise point the reader at `data/example/samples_manual_tpm.tsv`.

```
import pandas as pd

df = pd.read_csv("abundance/samples_manual_tpm.tsv", sep="\t")
print(df.sort_values("TPM", ascending=False).head(3))
```

Sorting by TPM highlights the most abundant viral operational taxonomic units. Save the resulting figures under `visualization/figures` to keep outputs consistent with the rest of the book.

9.6 Final mixed quiz

Note

Q1. Which tool estimates viral completeness and contamination? Answer: CheckV.
Q2. Which tool performs read trimming and quality filtering and creates an HTML report? Answer: fastp.
Q3. Which tool builds a maximum likelihood phylogenetic tree? Answer: IQ-TREE.
Q4. Which method can predict host association using bacterial immune memory? Answer: CRISPR spacer matching.
Q5. Which normalization can be used for viral abundance? Answer: TPM or coverage.

Estimated resources on 12 threads and 32 GB RAM: These practice tasks are lightweight and finish in seconds unless they reuse large FASTQ, BAM, or assembly files. The `data/example/` tables keep every command runnable on a laptop.

9.7 Key takeaways

- Work each exercise right after its matching chapter, then use the collapsible answers to test recall before revealing them.
- Treat every wrong quiz answer as a signal: reread the linked section rather than memorizing the correct letter.
- Practice mapping problems to tools — QC (fastp), quality (CheckV), taxonomy (vConTACT2), phylogenetics (IQ-TREE) — since exams often ask which tool solves which task.
- Rebuild the small `awk`, `cut`, `sort`, and `seqkit` one-liners from memory; fluency with tab-separated data is the most transferable skill.
- Self-assess with both the static and interactive quizzes, and only move on once you can explain each answer in your own words.

9.8 Further reading

- Chen et al. (2018) — revisit the QC step behind many practice exercises and confirm you can read a fastp report.
- Nayfach et al. (2021) — reread how completeness and contamination tiers are defined before self-testing on quality control.
- Minh et al. (2020) — review maximum likelihood tree building to answer the phylogenetics questions confidently.
- Quarto documentation on callouts and collapsible content: https://quarto.org/docs/authoring/callouts.html — useful if you want to build your own self-check boxes.

9.9 Chapter figure

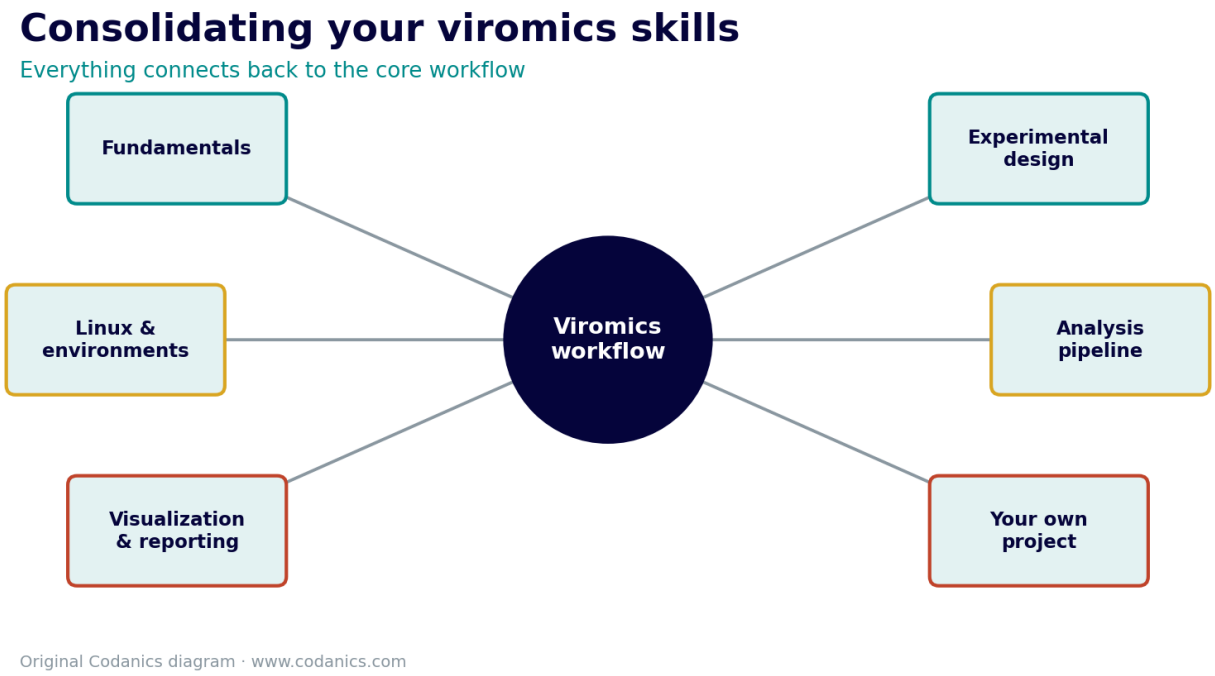


Figure 9.1: A skills consolidation map linking each practice topic to the stage of the viromics workflow it reinforces.

Generate this figure

Save as: `images/ch07-study-map.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean educational infographic titled "Skills consolidation map" showing the viromics workflow as a horizontal pipeline of connected stages, left to right: Fundamentals, Experimental design, Linux setup, Assembly and pipeline, Visualization and abundance. Draw each stage as a rounded card with a small icon (DNA strand, sample tubes, terminal window,

contig bars, bar chart). Above each card, place a small badge listing the skill practiced there (for example “awk filtering”, “metadata table”, “conda env check”, “seqkit length filter”, “TPM ranking”). Connect the cards with arrows to show progression, and add a circular “self-check quiz” node beneath the pipeline linking up to every stage. Use teal and navy Codanics branding, clear sans-serif labels, and a white background.

9.10 Interactive quiz: Mixed self-check

10 Project Ideas in Viromics

This chapter turns the skills from the book into concrete research directions. Each idea can grow into a short paper, an MSc project, a PhD chapter, or a postdoctoral pilot study. Every project names specific tools introduced earlier, so you can move from question to workflow without guessing which software to run.

Learning objectives — by the end of this chapter you will be able to:

- frame a focused, testable viromics research question tied to a sampling design;
- map a question onto a concrete tool chain from QC through host prediction;
- anticipate the compute and storage a project will need before you start; and
- outline a manuscript that reports viral recovery, quality, taxonomy, function, and abundance honestly.

10.1 Project 1: Soil fertilization and DNA virome shifts

Research question: How does nitrogen fertilization change soil phage diversity and vOTU abundance?

Design: soil DNA virome or total metagenome, treated versus control plots, 3 to 5 biological replicates per treatment.

Core workflow: fastp for QC and trimming, MEGAHIT for assembly, VirSorter2 and geNomad for viral identification, CheckV for quality, CD-HIT or vClust for vOTU clustering, CoverM for coverage, and diversity analysis in R.

Main outputs: a vOTU abundance matrix, CheckV quality tiers, viral taxonomy, and host prediction against soil MAGs or isolate genomes.

Estimated resources: 12 threads, 32 GB RAM, and 500 GB to 1 TB storage for 20 to 40 samples.

10.2 Project 2: Plant RNA virome in symptomatic and healthy leaves

Research question: Which RNA viruses are associated with disease symptoms in crop leaves?

Design: RNA extraction with rRNA depletion, cDNA sequencing, symptomatic versus healthy leaf pairs.

Core workflow: FastQC and fastp for quality control, an RNA-aware assembly, an RdRp marker search, viral classification with geNomad, and phylogenetics with IQ-TREE.

Main outputs: candidate plant RNA viruses, RdRp phylogenetic trees, and symptom-associated abundance.

Estimated resources: 12 threads, 32 GB RAM, and 300 to 800 GB storage depending on sample count.

10.3 Project 3: Wastewater viral surveillance

Research question: Can wastewater metagenomics recover seasonal viral signals?

Design: repeated wastewater sampling across weeks or months at one or more sites.

Core workflow: SRA or local FASTQ retrieval, fastp trimming, MEGAHIT assembly, viral identification with VirSorter2 and geNomad, CheckV quality control, and abundance mapping with Bowtie2 and CoverM.

Main outputs: viral diversity over time, enteric virus signals, and phage community structure.

Estimated resources: 12 threads, 32 GB RAM, and 1 TB or more for longitudinal studies.

10.4 Project 4: Host prediction benchmarking

Research question: How consistent are iPHoP, WIsH, CRISPR spacer matching, and taxonomy-based host predictions?

Design: public phage genomes with known hosts plus environmental viral contigs as a novel test set.

Core workflow: run iPHoP, WIsH, and CRISPR spacer matching, then build an agreement matrix and measure precision where the true host is known.

Main outputs: a method comparison table, confidence classes, and a practical host prediction decision framework.

Estimated resources: 12 threads, 32 GB RAM, and 200 to 600 GB, driven mostly by host databases.

10.5 Project 5: Auxiliary metabolic genes in soil or marine viromes

Research question: Which viral auxiliary metabolic genes are linked to nutrient cycling?

Design: soil or marine virome assemblies from an ecosystem of interest.

Core workflow: viral prediction with VirSorter2 and geNomad, CheckV quality control, functional annotation with DRAM-v and VIBRANT, orthology with eggNOG-mapper, and careful AMG filtering.

Main outputs: AMG functional categories, viral contig quality, and an ecosystem-function interpretation.

Estimated resources: 12 threads, 32 GB RAM, and 500 GB to 1 TB with full functional databases.

10.6 Project 6: Viral dark matter in a local ecosystem

Research question: What fraction of predicted viral contigs stays unclassified after multiple tools?

Design: any environmental metagenome you can access, ideally with matched metadata.

Core workflow: VirSorter2 and geNomad for identification, CheckV for quality, vConTACT2 and PhaBOX2 for taxonomy, then a comparison that summarizes unclassified vOTUs.

Main outputs: classified versus unclassified viral fractions, a novelty estimate, and candidate viral clusters worth follow-up.

Estimated resources: 12 threads, 32 GB RAM, and 300 GB to 1 TB.

10.7 Suggested paper outline

```
Title
Abstract
Introduction
Materials and Methods
Results
  Read quality and assembly
  Viral contig recovery
  CheckV quality
  vOTU clustering
  Taxonomic classification
  Functional annotation
  Abundance and diversity
  Host prediction
Discussion
Limitations
Conclusion
Data and code availability
References
```

10.8 Key takeaways

- Start every project from a focused, testable question tied to a concrete sampling design and a matching tool chain.
- Scope the study to your compute and storage budget before you commit — replicates, controls, and read depth all drive feasibility.

- Reuse standard building blocks (QC, assembly, viral identification, CheckV quality, clustering, abundance, host prediction) rather than reinventing each pipeline.
- Report viral recovery, quality tiers, taxonomy, function, and abundance honestly, and always include a limitations section.
- Public data lowers the barrier to entry: an SRA-based reanalysis can become a solid first paper without new sequencing.

10.9 Further reading

- Camargo et al. (2024) and Nayfach et al. (2021) — the identification-plus-quality core that anchors most project workflows.
- Roux et al. (2023) — for scoping any project with a host prediction component.
- International Committee on Taxonomy of Viruses (2026) — the reference framework for reporting and interpreting viral taxonomy in a manuscript.
- NCBI SRA for finding public datasets and depositing your own reads: https://www.ncbi.nlm.nih.gov/sra.

10.10 Chapter figure

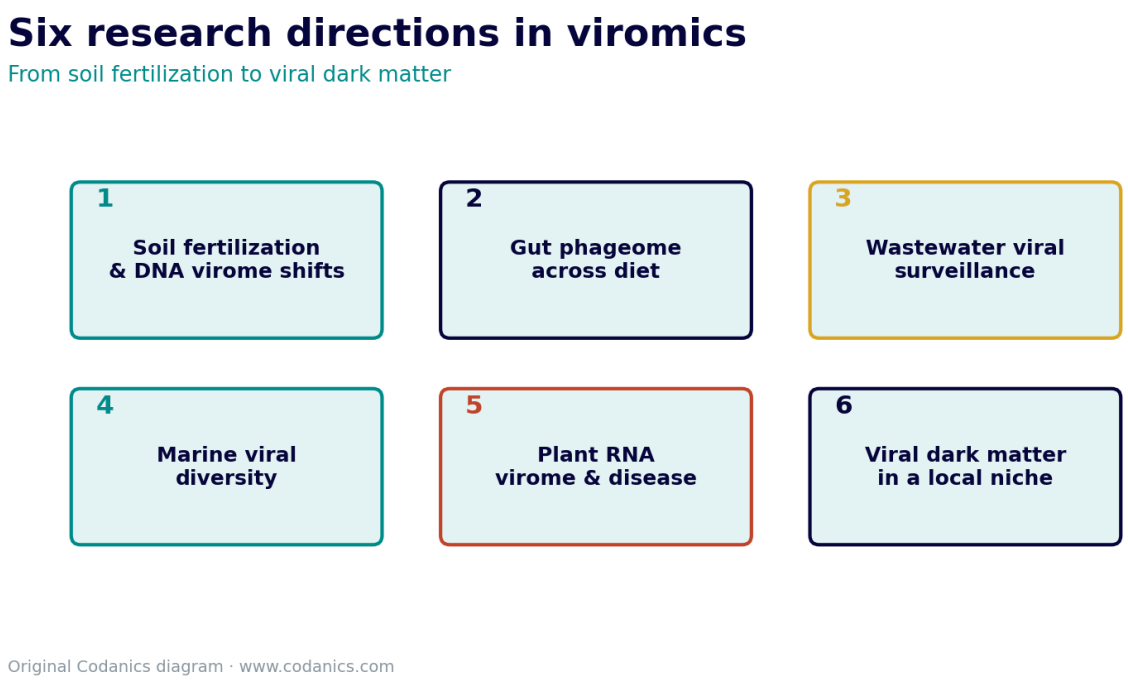


Figure 10.1: Six viromics project ideas, each pairing a research question with its core tool chain.

Generate this figure

Save as: `images/ch08-project-ideas.png` · **Aspect ratio:** 16:9 · **Style:** clean flat vector infographic, Codanics palette (teal `#008b8b`, navy `#05043b`, white background), no photorealism.
Prompt: Create a clean educational infographic laid out as a two-by-three grid of six project cards, each with a simple icon and short label: (1) Soil fertilization DNA virome — soil layers with a fertilizer bag and phage icons; (2) Plant RNA virome — a crop leaf split into healthy and symptomatic halves with virus particles; (3) Wastewater surveillance — a water treatment pipe with a calendar timeline; (4) Host prediction benchmarking — phage particles connecting to bacterial cells with a comparison grid; (5) Auxiliary metabolic genes — a viral contig with highlighted metabolic gene boxes; (6) Viral dark matter — a cluster of unknown contigs with question marks. Give each card a small strip of tool names underneath (for example "VirSorter2 · CheckV · CoverM"). Use teal and navy Codanics branding, clear sans-serif labels, and a white background.

10.11 Quiz: Project Ideas

Q1. Which project is best for crop disease discovery?

A. plant RNA virome B. Bowtie2 benchmarking only C. PDF formatting D. SRA download testing only

Note

Answer: A. Plant RNA viromes are well suited for viral disease discovery.

Q2. Which project focuses on AMGs?

A. auxiliary metabolic genes in viromes B. metadata formatting only C. FastQC comparison only D. PDF rendering

Note

Answer: A. DRAM-v and VIBRANT can help interpret viral functional genes.

Q3. Which project needs host genomes or MAGs most strongly?

A. host prediction benchmarking B. dark mode design C. read compression only D. chapter writing

Note

Answer: A. Host prediction depends on relevant host reference data.

Q4. Which design is best for surveillance?

A. longitudinal wastewater sampling B. one blank only C. one unknown file D. no metadata

Note

Answer: A. Surveillance benefits from repeated sampling over time.

Q5. What section should mention limitations?

A. Discussion B. FASTQ filename only C. title D. logo

Note

Answer: A. Viromics studies need transparent discussion of database and prediction limits.

10.12 Interactive quiz: Project ideas

References

Baltimore, David. 1971. "Expression of Animal Virus Genomes." *Bacteriological Reviews* 35 (3): 235–41. https://doi.org/10.1128/br.35.3.235-241.1971.

Bin Jang, Ho, Benjamin Bolduc, Olivier Zablocki, Jens H. Kuhn, Simon Roux, Evelien M. Adriaenssens, J. Rodney Brister, et al. 2019. "Taxonomic Assignment of Uncultivated Prokaryotic Virus Genomes Is Enabled by Gene-Sharing Networks." *Nature Biotechnology* 37: 632–39. https://doi.org/10.1038/s41587-019-0100-8.

Bolger, Anthony M., Marc Lohse, and Bjoern Usadel. 2014. "Trimmomatic: A Flexible Trimmer for Illumina Sequence Data." *Bioinformatics* 30 (15): 2114–20. https://doi.org/10.1093/bioinformatics/btu170.

Camargo, Antonio Pedro, Simon Roux, Frederik Schulz, Michal Babinski, Yan Xu, Bin Hu, Patrick S. G. Chain, Stephen Nayfach, and Nikos C. Kyrpides. 2024. "Identification of Mobile Genetic Elements with geNomad." *Nature Biotechnology* 42: 1303–12. https://doi.org/10.1038/s41587-023-01953-y.

Cantalapiedra, Carlos P., Ana Hernandez-Plaza, Ivica Letunic, Peer Bork, and Jaime Huerta-Cepas. 2021. "eggNOG-Mapper V2: Functional Annotation, Orthology Assignments, and Domain Prediction at the Metagenomic Scale." *Molecular Biology and Evolution* 38 (12): 5825–29. https://doi.org/10.1093/molbev/msab293.

Chen, Shifu, Yanqing Zhou, Yaru Chen, and Jia Gu. 2018. "Fastp: An Ultra-Fast All-in-One FASTQ Preprocessor." *Bioinformatics* 34 (17): i884–90. https://doi.org/10.1093/bioinformatics/bty560.

Danecek, Petr, James K. Bonfield, Jennifer Liddle, John Marshall, Valeriu Ohan, Martin O. Pollard, Andrew Whitwham, et al. 2021. "Twelve Years of SAMtools and BCFtools." *GigaScience* 10 (2): giab008. https://doi.org/10.1093/gigascience/giab008.

Fu, Limin, Beifang Niu, Zhengwei Zhu, Sitao Wu, and Weizhong Li. 2012. "CD-HIT: Accelerated for Clustering the Next-Generation Sequencing Data." *Bioinformatics* 28 (23): 3150–52. https://doi.org/10.1093/bioinformatics/bts565.

Galiez, Clément, Matthias Siebert, François Enault, Jonathan Vincent, and Johannes Söding. 2017. "WIsH: Who Is the Host? Predicting Prokaryotic Hosts from Metagenomic Phage Contigs." *Bioinformatics* 33 (19): 3113–14. https://doi.org/10.1093/bioinformatics/btx383.

Guo, Jiarong, Benjamin Bolduc, Ahmed A. Zayed, Arvind Varsani, Gabriela Dominguez-Huerta, Tom O. Delmont, Akbar A. Pratama, et al. 2021. "VirSorter2: A Multi-Classifier, Expert-Guided Approach to Detect Diverse DNA and RNA Viruses." *Microbiome* 9: 37. https://doi.org/10.1186/s40168-020-00990-y.

Gurevich, Alexey, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. 2013. "QUAST: Quality Assessment Tool for Genome Assemblies." *Bioinformatics* 29 (8): 1072–75. https://doi.org/10.1093/bioinformatics/btt086.

Hyatt, Doug, Gwo-Liang Chen, Philip F. LoCascio, Miriam L. Land, Frank W. Larimer, and Loren J. Hauser. 2010. "Prodigal: Prokaryotic Gene Recognition and Translation Initiation Site Identification." *BMC Bioinformatics* 11: 119. https://doi.org/10.1186/1471-2105-11-119.

International Committee on Taxonomy of Viruses. 2026. "ICTV Taxonomy." https://ictv.global/taxonomy.

Katoh, Kazutaka, and Daron M. Standley. 2013. "MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability." *Molecular Biology and Evolution* 30 (4): 772–80. https://doi.org/10.1093/molbev/mst010.

Kieft, Kristopher, Zhichao Zhou, and Karthik Anantharaman. 2020. "VIBRANT: Automated Recovery, Annotation and Curation of Microbial Viruses, and Evaluation of Viral Community Function from Genomic Sequences." *Microbiome* 8: 90. https://doi.org/10.1186/s40168-020-00867-0.

Langmead, Ben, and Steven L. Salzberg. 2012. "Fast Gapped-Read Alignment with Bowtie 2." *Nature Methods* 9 (4): 357–59. https://doi.org/10.1038/nmeth.1923.

Li, Dinghua, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. 2015. "MEGAHIT: An Ultra-Fast Single-Node Solution for Large and Complex Metagenomics Assembly via Succinct de Bruijn Graph." *Bioinformatics* 31 (10): 1674–76. https://doi.org/10.1093/bioinformatics/btv033.

Minh, Bui Quang, Heiko A. Schmidt, Olga Chernomor, Dominik Schrempf, Michael D. Woodhams, Arndt von Haeseler, and Robert Lanfear. 2020. "IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era." *Molecular Biology and Evolution* 37 (5): 1530–34. https://doi.org/10.1093/molbev/msaa015.

Minot, Samuel, Rohini Sinha, Jun Chen, Hongzhe Li, Sue A. Keilbaugh, Gary D. Wu, James D. Lewis, and Frederic D. Bushman. 2011. "The Human Gut Virome: Inter-Individual Variation and Dynamic Response to Diet." *Genome Research* 21 (10): 1616–25. https://doi.org/10.1101/gr.122705.111.

Nayfach, Stephen, Antonio Pedro Camargo, Frederik Schulz, Emiley Eloe-Fadrosh, Simon Roux, and Nikos C. Kyrpides. 2021. "CheckV Assesses the Quality and Completeness of Metagenome-Assembled Viral Genomes." *Nature Biotechnology* 39: 578–85. https://doi.org/10.1038/s41587-020-00774-7.

NCBI. 2026. "Sequence Read Archive." https://www.ncbi.nlm.nih.gov/sra.

NCBI Sequence Read Archive. 2022. "Download SRA Sequences from Entrez Search Results." https://www.ncbi.nlm.nih.gov/sra/docs/sradownload.

———. 2024. "SRA Run SRR29680455, BioProject PRJNA527877, Experiment SRX25183710." https://www.ncbi.nlm.nih.gov/sra.

Nurk, Sergey, Dmitry Meleshko, Anton Korobeynikov, and Pavel A. Pevzner. 2017. "metaSPAdes: A New Versatile Metagenomic Assembler." *Genome Research* 27 (5): 824–34. https://doi.org/10.1101/gr.213959.116.

Posit. 2024. "Quarto Documentation." https://quarto.org/docs/books/.

Ren, Jie, Nathan A. Ahlgren, Yang Young Lu, Jed A. Fuhrman, and Fengzhu Sun. 2017. "VirFinder: A Novel k-Mer Based Tool for Identifying Viral Sequences from Assembled Metagenomic Data." *Microbiome* 5: 69. https://doi.org/10.1186/s40168-017-0283-5.

Ren, Jie, Kai Song, Chao Deng, Nathan A. Ahlgren, Jed A. Fuhrman, Yi Li, Xiaohui Xie, Ryan Poplin, and Fengzhu Sun. 2020. "Identifying Viruses from Metagenomic Data Using Deep Learning." *Quantitative Biology* 8 (1): 64–77. https://doi.org/10.1007/s40484-019-0187-4.

Roux, Simon, Evelien M. Adriaenssens, Bas E. Dutilh, Eugene V. Koonin, Andrew M. Kropinski, Mart Krupovic, Jens H. Kuhn, et al. 2019. "Minimum Information about an Uncultivated Virus Genome (MIUViG)." *Nature Biotechnology* 37: 29–37. https://doi.org/10.1038/nbt.4306.

Roux, Simon, Antonio Pedro Camargo, Felipe H. Coutinho, Shareef M. Dabdoub, Bas E. Dutilh, et al. 2023. "iPHoP: An Integrated Machine Learning Framework to Maximize Host Prediction for Metagenome-Derived Viruses of Archaea and Bacteria." *PLOS Biology* 21 (4): e3002083.

https://doi.org/10.1371/journal.pbio.3002083.

Roux, Simon, Francois Enault, Bonnie L. Hurwitz, and Matthew B. Sullivan. 2015. "VirSorter: Mining Viral Signal from Microbial Genomic Data." *PeerJ* 3: e985. https://doi.org/10.7717/peerj.985.

Shaffer, Michael, Mikayla A. Borton, Brendan B. McGivern, Ahmed A. Zayed, Sabina L. La Rosa, Lindsey M. Solden, Pengfei Liu, et al. 2020. "DRAM for Distilling Microbial Metabolism to Automate the Curation of Microbiome Function." *Nucleic Acids Research* 48 (16): 8883–8900. https://doi.org/10.1093/nar/gkaa621.

Shang, Jiayu, Cheng Peng, Jiaojiao Guan, Dehan Cai, Donglin Wang, and Yanni Sun. 2026. "PhaBOX2: An Enhanced Web Server for Discovering and Analyzing Viral Contigs in Metagenomic Data." *Nucleic Acids Research* 54: W169–76. https://doi.org/10.1093/nar/gkag382.

Shen, Wei, Shuai Le, Yan Li, and Fuquan Hu. 2016. "SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/q File Manipulation." *PLOS ONE* 11 (10): e0163962. https://doi.org/10.1371/journal.pone.0163962.

Zielezinski, Andrzej, Adam Gudyś, Jakub Barylski, Krzysztof Siminski, Piotr Rozwalak, Bas E. Dutilh, and Sebastian Deorowicz. 2025. "Ultrafast and Accurate Sequence Alignment and Clustering of Viral Genomes." *Nature Methods* 22: 1191–94. https://doi.org/10.1038/s41592-025-02701-7.

A Linux Cheatsheet for Viromics

A scannable quick reference for the commands used most often across this book. Copy, adapt, and keep it open in a second terminal. Commands are grouped by task so you can jump straight to what you need.

> The one habit that saves the most time
>
> Preview before you overwrite or delete. Run a `ls`, `head`, or a `--dry-run` first, and only then commit to the real command. Most lost-data disasters come from a hasty `rm` or a redirect (`>`) onto the wrong file.

A.1 Navigation

```
pwd                          # where am I
ls -lh                       # list with human-readable sizes
ls -lhtr                     # sort by time, newest last (good for logs)
cd ~/viromics_course         # jump to the project root
cd -                         # jump back to the previous directory
tree -L 2                    # show the folder tree, two levels deep
du -sh *                     # size of everything in the current folder
```

A.2 Files and folders

```
mkdir -p raw_reads trimmed_reads reports   # -p makes parents, no error if present
cp -r results results_backup               # -r copies folders
mv old_name new_name                       # rename or move
rm file                                    # delete a file (no undo)
rm -r folder                               # delete a folder and its contents
ln -s /big/disk/db databases               # symlink a large DB into the project
find . -name "*.fastq.gz" -size +1G        # find large FASTQ files
```

A.3 Viewing and inspecting

```
head -n 20 file.tsv          # first 20 lines
tail -n 20 file.tsv          # last 20 lines
tail -f logs/run.log         # follow a log live (Ctrl-C to stop)
less -S table.tsv            # page a wide table without wrapping (q to quit)
wc -l file.tsv               # count lines
zcat reads.fastq.gz | head   # peek inside a gzip file without unzipping
column -t -s $'\t' file.tsv  # pretty-print a TSV
```

A.4 Text processing (grep, awk, sed, cut, sort, uniq)

```
grep -c "^>" contigs.fa                    # count sequences in a FASTA
grep -v "^#" table.tsv                     # drop comment/header lines
grep -i "virsorter" logs/*.log             # case-insensitive search across logs

cut -f1,3 table.tsv                        # keep columns 1 and 3
cut -f1 table.tsv | tail -n +2             # column 1, minus the header

sort -k2,2 -t$'\t' table.tsv               # sort by column 2 (tab-separated)
sort -k5,5nr table.tsv                     # sort by column 5, numeric, descending
sort file.txt | uniq -c | sort -nr         # count and rank unique values

awk -F'\t' 'NR>1 && $4=="High-quality"' checkv.tsv   # filter rows by a column value
awk -F'\t' '$2>=10000' lengths.tsv                   # keep contigs >= 10 kb
awk -F'\t' '{sum+=$2} END{print sum}' lengths.tsv    # sum a column

sed 's/old/new/g' file.txt                 # replace text (prints to screen)
sed -i 's/old/new/g' file.txt              # replace in place (edits the file)
sed 's/ .*//' contigs.fa                   # keep only the first word of FASTA headers
tr ',' '\t' < file.csv > file.tsv          # convert commas to tabs
paste a.txt b.txt                          # merge two files column-wise
```

A.5 FASTA and FASTQ with seqkit

```
seqkit stats *.fastq.gz                              # N, length stats, N50
seqkit seq -m 1000 contigs.fa > contigs_min1kb.fa    # keep sequences >= 1 kb
seqkit grep -f ids.txt contigs.fa > selected.fa      # extract by ID list
seqkit grep -v -f drop.txt contigs.fa > kept.fa      # exclude an ID list
seqkit fx2tab -n -l contigs.fa > lengths.tsv         # name + length table
seqkit rmdup -s contigs.fa > dedup.fa                # remove exact duplicates
seqkit subseq -r 1:5000 genome.fa                    # slice a region
seqkit sort -l -r contigs.fa > by_length.fa          # sort by length, longest first
seqkit replace -p ".*" -r "ctg_{nr}" contigs.fa      # renumber headers cleanly
```

A.6 Conda and mamba

```
conda env list                                              # list environments
mamba create -n viromics-core -c conda-forge -c bioconda fastqc fastp multiqc
conda activate viromics-core
mamba install -c conda-forge -c bioconda seqkit             # add a tool to the active env
conda deactivate
conda env export --no-builds > env.yml                      # record an environment
mamba env create -f env.yml                                 # rebuild it elsewhere
conda config --set channel_priority strict                  # avoids many solver conflicts
```

Use `mamba`, and one job per environment

`mamba` resolves dependencies far faster than the classic conda solver. When two heavy tools conflict, do not fight the solver — give each its own environment (`virsorter2`, `genomad`, `checkv`, …) and switch between them.

A.7 Bash scripting

```
cat > run.sh << 'EOF'
#!/usr/bin/env bash
set -euo pipefail                  # stop on errors, unset vars, and failed pipes
THREADS=12
for r1 in trimmed_reads/*_R1.fastq.gz; do
  sample=$(basename "$r1" _R1.fastq.gz)
  echo ">> processing $sample"
  # your command here, using "$sample" and "$THREADS"
done
EOF

chmod +x run.sh
bash run.sh 2>&1 | tee logs/run.log     # run and save all output to a log
```

Handy building blocks:

```
VAR="value"; echo "$VAR"                # set and use a variable
export VIROME_DB=~/databases            # export so child processes see it
sample=$(basename "$path" .fastq.gz)    # strip a directory and suffix
for f in *.fa; do echo "$f"; done       # loop over files
cmd1 && cmd2                            # run cmd2 only if cmd1 succeeds
cmd1 || echo "cmd1 failed"              # fallback on failure
```

A.8 Long-running jobs with tmux and screen

Assemblies, database downloads, and prediction runs can take hours. Detach them so a dropped SSH connection does not kill the job.

```
tmux new -s viromics        # start a named session
#   ... launch your long command inside it ...
# Detach: press Ctrl-b then d       (the job keeps running)
tmux ls                     # list sessions
tmux attach -t viromics     # reattach later

# screen equivalents:
screen -S viromics          # start
# Detach: Ctrl-a then d
screen -ls                  # list
screen -r viromics          # reattach

# Or run detached and log to a file:
nohup bash run_phase4_main_pipeline.sh > logs/phase4.log 2>&1 &
```

A.9 Monitoring resources

```
htop                        # live CPU/RAM per process (q to quit; F9 to kill)
top                         # always-available fallback for htop
df -h                       # free disk space per filesystem
du -sh ~/viromics_course/*  # which project folders are eating disk
du -sh databases/*          # database sizes
free -h                     # free and used RAM
nproc                       # how many CPU threads are available
ls -lh out/*.bam            # check output file sizes as a job progresses
```

> ⚠ Watch disk during assembly
>
> Assembly and mapping create large temporary files. Run `df -h` before you start a big job, and again if it fails with a write error — a full disk is the most common silent cause of a crashed assembly.

B Viromics Tool Summary Table

A one-stop reference for every tool used in this book, plus close alternatives you may meet in the literature. Install commands use the same channels as Chapter 3 — always `-c conda-forge -c bioconda` — so environments resolve consistently. When two heavy tools conflict, install each in its own conda environment rather than forcing them together.

> **Reading the install column**
>
> Every `mamba install` below assumes you have already added the channels shown, or that you pass them each time. A safe, copy-paste-ready form is:
>
> ```
> mamba create -n <env> -c conda-forge -c bioconda <package>
> ```
>
> Tools without a conda package (DeepVirFinder, WIsH) are installed from source or from the authors' repository, as noted.

B.1 Full tool table

Tool	Category	Purpose	Input	Output	Install	Notes
FastQC	QC	per-sample read quality reports	FASTQ	HTML, ZIP	`mamba install -c conda-forge -c bioconda fastqc`	first QC step
MultiQC	QC	combine many reports into one	report folders	HTML	`mamba install -c conda-forge -c bioconda multiqc`	run after FastQC across samples

Tool	Category	Purpose	Input	Output	Install	Notes
fastp	trimming	trim adapters and low-quality bases	FASTQ	cleaned FASTQ, HTML	`mamba install -c conda-forge -c bioconda fastp`	fast all-in-one preprocessor
Trimmomatic	trimming	alternative read trimmer	FASTQ	paired/unpaired FASTQ	`mamba install -c conda-forge -c bioconda trimmomatic`	useful for teaching comparisons
MEGAHIT	assembly	metagenomic assembly	clean FASTQ	contigs	`mamba install -c conda-forge -c bioconda megahit`	fast, memory efficient; good on 32 GB RAM
metaSPAdes	assembly	metagenomic assembly	clean FASTQ	contigs, scaffolds	`mamba install -c conda-forge -c bioconda spades`	higher quality, heavier than MEGAHIT
QUAST	assembly QC	assembly statistics	contigs	reports	`mamba install -c conda-forge -c bioconda quast`	reports N50, length, GC; N50 alone is not enough
Prodigal	gene calling	predict protein-coding ORFs	contigs	FAA, GFF	`mamba install -c conda-forge -c bioconda prodigal`	use `-p meta`; feeds protein-based tools (Hyatt et al. 2010)

Tool	Category	Purpose	Input	Output	Install	Notes
VirSorter2	viral prediction	identify viral contigs	contigs	viral FASTA, scores	`mamba install -c conda-forge -c bioconda virsorter=2`	multi-classifier; pair with CheckV (Guo et al. 2021)
geNomad	viral prediction	identify viruses and plasmids	contigs	predictions, taxonomy	`mamba install -c conda-forge -c bioconda genomad`	strong modern default; end-to-end mode (Camargo et al. 2024)
VirFinder	viral prediction	k-mer model for viral contigs	contigs	scores, p-values	`mamba install -c conda-forge -c bioconda r-virfinder`	reference-free; R package (Ren et al. 2017)
DeepVirFinder	viral prediction	deep-learning viral scoring	contigs	scores, p-values	install from source (GitHub)	successor to VirFinder; no conda package (Ren et al. 2020)
VIBRANT	viral prediction/annotation	recover and annotate viruses	contigs	viral FASTA, annotations	`mamba install -c conda-forge -c bioconda vibrant`	flags AMGs; microbial viruses
CheckV	quality	completeness and contamination	viral FASTA	quality tables, trimmed FASTA	`mamba install -c conda-forge -c bioconda checkv`	essential QC filter; trims host flanks (Nayfach et al. 2021)

Tool	Category	Purpose	Input	Output	Install	Notes
CD-HIT	clustering	fast sequence dereplication	FASTA	representatives, clusters	`mamba install -c conda-forge -c bioconda cd-hit`	simple and fast; `cd-hit-est` for nucleotides (Fu et al. 2012)
vClust	clustering	ANI-based vOTU clustering	viral FASTA	ANI table, clusters	`mamba install -c conda-forge -c bioconda vclust`	95% ANI vOTUs; add `--qcov 0.85` for the MIUViG alignment-fraction cutoff (Zielezinski et al. 2025)
vConTACT2	taxonomy	gene-sharing network taxonomy	proteins, gene map	viral clusters (VCs)	`mamba install -c conda-forge -c bioconda vcontact2`	prokaryotic viruses (Bin Jang et al. 2019)
PhaBOX2 / PhaGCN	taxonomy/lifestyle	phage taxonomy, lifestyle, host	viral FASTA	taxonomy, lifestyle, host	`mamba create -n phabox -c conda-forge -c bioconda phabox`	integrated phage toolkit; graph + learning models (Shang et al. 2026)
DRAM-v	annotation	viral functions and AMGs	VirSorter2 output	annotation tables	install in a dedicated DRAM env	database heavy; distilled AMG summary (Shaffer et al. 2020)

Tool	Category	Purpose	Input	Output	Install	Notes
eggNOG-mapper	annotation	orthology and function	proteins	annotation table	`mamba install -c conda-forge -c bioconda eggnog-mapper`	broad functional annotation (Cantalapiedra et al. 2021)
Bowtie2	mapping	align reads to vOTUs	reads, index	SAM/BAM	`mamba install -c conda-forge -c bioconda bowtie2`	abundance step; build index first (Langmead and Salzberg 2012)
samtools	BAM handling	sort, index, filter alignments	SAM/BAM	sorted, indexed BAM	`mamba install -c conda-forge -c bioconda samtools`	glue for the mapping workflow (Danecek et al. 2021)
CoverM	coverage	per-contig coverage and TPM	BAM or FASTQ	coverage/abundance table	`mamba install -c conda-forge -c bioconda coverm`	builds the vOTU abundance table
iPHoP	host prediction	integrated host prediction	viral FASTA	host predictions	`mamba install -c conda-forge -c bioconda iphop`	database heavy; combines multiple signals
WIsH	host prediction	composition-based host prediction	host + viral FASTA	log-likelihood scores	install from source (GitHub)	needs a relevant host genome set (Galiez et al. 2017)

Tool	Category	Purpose	Input	Output	Install	Notes
MAFFT	phylogeny	multiple sequence alignment	marker FASTA	aligned FASTA	`mamba install -c conda-forge -c bioconda mafft`	align homologous markers (Katoh and Standley 2013)
IQ-TREE 2	phylogeny	maximum-likelihood trees	alignment	tree, support values	`mamba install -c conda-forge -c bioconda iqtree`	model selection + bootstraps (Minh et al. 2020)

B.2 Choosing between overlapping tools

If you need to…	Reasonable default	Alternative	Why
Call viral contigs	geNomad	VirSorter2	geNomad is fast and current; run both for agreement
Score novel/short contigs	DeepVirFinder	VirFinder	reference-free k-mer/deep-learning signal fills reference gaps
Assemble on 32 GB RAM	MEGAHIT	metaSPAdes	MEGAHIT is lighter; metaSPAdes is higher quality when RAM allows
Dereplicate into vOTUs	vClust	CD-HIT	vClust clusters by ANI; pass `--qcov 0.85` with `--ani 0.95` for the MIUViG 85% alignment-fraction rule
Predict host	iPHoP	WIsH	iPHoP integrates many signals; WIsH is lightweight but needs candidate hosts
Assign phage taxonomy	geNomad	vConTACT2 / PhaBOX	geNomad is quick; the others add network and graph evidence

C Troubleshooting Guide

Practical fixes for the errors that stop viromics pipelines most often. Each entry follows a **Problem → Fix** format. Start at the top of the list that matches your symptom, and change one thing at a time.

> **Read the last 20 lines of the log first**
>
> Most failures announce themselves. Before changing anything, run `tail -n 20 logs/<step>.log` (or `tail -f` for a live job). The real error is usually near the end, not in the wall of warnings above it.

C.1 `command not found`

Problem: the shell cannot find a tool — usually the wrong environment is active, or the tool was never installed.

Fix:

```
conda env list                          # which environments exist
conda activate viromics-core            # activate the right one
which fastqc                             # confirm the tool is on PATH
```

If `which` finds nothing, install the tool into that environment.

C.2 Conda solver is too slow or hangs

Problem: `conda install` spins for many minutes on "Solving environment".

Fix: use `mamba` (or the libmamba solver) and strict channel priority.

```
conda install -n base -c conda-forge mamba          # one-time
mamba install -c conda-forge -c bioconda checkv     # use mamba from now on

# Or keep conda but switch its solver:
conda config --set solver libmamba
conda config --set channel_priority strict
```

C.3 Conda environment conflicts

Problem: two tools demand incompatible dependencies and refuse to co-install.

Fix: give each heavy tool its own environment instead of forcing them together.

```
mamba create -n checkv    -c conda-forge -c bioconda checkv
mamba create -n virsorter2 -c conda-forge -c bioconda virsorter=2
```

C.4 SRA download is slow or `prefetch` fails

Problem: `prefetch`/`fasterq-dump` stalls, times out, or errors on a public accession.

Fix: update the SRA Toolkit, run `vdb-config` once, prefetch before dumping, and raise the size cap. Retry — public mirrors are sometimes briefly unavailable.

```
mamba install -c conda-forge -c bioconda sra-tools
vdb-config --interactive          # accept defaults, set a cache dir once

prefetch --max-size 100g SRR000000                # download the .sra first
fasterq-dump --split-files -e 12 -O raw_reads SRR000000
gzip raw_reads/SRR000000_*.fastq

# If prefetch keeps failing, fall back to a direct HTTPS/FTP mirror (ENA)
# or add:  --transport http
```

C.5 geNomad or CheckV database version mismatch

Problem: a run errors on the database, or results look wrong, because the installed tool version expects a different database version than the one on disk.

Fix: match the database to the tool. Re-download the database with the same tool version you are running, and point the tool at the exact folder.

```
genomad --version
genomad download-database $VIROME_DB/genomad        # re-fetch for this version
ls $VIROME_DB/genomad/genomad_db                     # confirm the versioned folder

checkv --version
# Re-download CheckV DB if the tool was upgraded:
checkv download_database $VIROME_DB/checkv
export CHECKVDB=$VIROME_DB/checkv/checkv-db-vX.Y      # set to the actual folder name
```

Keep the tool and its database in the **same** conda environment so upgrades stay in sync.

C.6 Database not found (CheckV / geNomad path errors)

Problem: the tool cannot locate its database directory.

Fix: confirm the environment variable and point to the folder that actually contains the DB.

```
echo $CHECKVDB
find $VIROME_DB/checkv  -maxdepth 2 -type d          # find the real checkv-db dir
find $VIROME_DB/genomad -maxdepth 2 -type d          # use the folder holding genomad_db
```

C.7 Disk full during assembly

Problem: assembly or mapping crashes with a write/"No space left on device" error. Assemblers create large temporary files.

Fix: check free space first, then free space, move temp to a bigger disk, and cap contig length.

```
df -h                                # find the full filesystem
du -sh ~/viromics_course/* databases/*   # find the big consumers

# Point MEGAHIT/temp at a larger disk and clean intermediates:
megahit -1 R1.fastq.gz -2 R2.fastq.gz -o /big/disk/out \
        --tmp-dir /big/disk/tmp --min-contig-len 1000 -t 12
rm -r out/intermediate_contigs        # remove after a successful run
```

C.8 Low memory during assembly

Problem: metaSPAdes exhausts RAM on a 32 GB machine.

Fix: switch to MEGAHIT, which is far more memory-efficient.

```
megahit -1 R1.fastq.gz -2 R2.fastq.gz -o out --min-contig-len 1000 -t 12
```

C.9 No contigs after assembly

Problem: the assembler finishes but produces an empty or tiny contig file.

Fix: check that reads survived trimming and read the assembler log.

```
seqkit stats trimmed_reads/*.fastq.gz     # are there reads left?
tail logs/samples_megahit.log             # what did the assembler say
```

C.10 Empty viral output

Problem: the viral prediction step returns no (or almost no) sequences.

Fix: the usual causes are biological or threshold-related:

- the dataset genuinely has few viral reads;
- contigs are too short to score confidently;
- the score cutoff is too strict;
- the database is missing or mismatched;
- the assembly is highly fragmented.

Lower the minimum length for exploration, confirm the database, then return to stricter criteria for the final report.

C.11 Empty vOTU file

Problem: the vOTU FASTA has no sequences.

Fix: confirm the upstream viral FASTA exists and is non-empty.

```
seqkit stats $VIRAL_FASTA
grep -c "^>" $VIRAL_FASTA
```

C.12 Bowtie2 index error

Problem: mapping fails because the index is missing or the path is wrong.

Fix: rebuild the index and point Bowtie2 at the correct prefix.

```
bowtie2-build $VOTU_FASTA abundance/index/sample_votus
bowtie2 -x abundance/index/sample_votus -1 R1 -2 R2 -S out.sam
```

C.13 Figures script: R package not installed

Problem: `make_figures.R` (or another R step) stops with `there is no package called '...'`.

Fix: install the missing R packages into the environment you run R from, then re-run.

```
mamba install -n viromics-core -c conda-forge \
  r-ggplot2 r-pheatmap r-vegan r-dplyr r-readr r-tidyr

# Or from inside R for a one-off:
Rscript -e 'install.packages("pheatmap", repos="https://cloud.r-project.org")'
```

C.14 Python module missing

Problem: a Python figure or TPM script fails with `ModuleNotFoundError`.

Fix: install into the active environment.

```
mamba install -n viromics-core -c conda-forge pandas matplotlib numpy seaborn
```

D Glossary

Concise, one-line definitions of the terms used throughout this book. Terms are listed alphabetically so you can scan quickly.

Term	Meaning
Alpha diversity	within-sample diversity (richness and evenness of viruses in one sample)
AMG (auxiliary metabolic gene)	host-derived metabolic gene carried by a virus that can reprogram host metabolism during infection
ANI (average nucleotide identity)	average identity between two genomes over aligned regions; species-level vOTUs use 95% ANI **over an 85% alignment fraction** (MIUViG)
CheckV contamination	in CheckV, host DNA flanking an integrated provirus (not cross-sample contamination); it is trimmed off and is **not** used to lower the quality tier
Baltimore classification	grouping of viruses (I–VII) by how their genome produces mRNA
Bacteriophage	virus that infects bacteria
Beta diversity	between-sample difference in viral community composition
Capsid	protein shell that encloses and protects a viral genome
CheckV completeness	estimated fraction of a viral genome recovered in a contig
Contig	a contiguous sequence assembled from overlapping reads
Coverage / depth	number of reads aligned across a position or contig; reflects abundance
CRISPR spacer	short host-derived sequence in a CRISPR array that can match and identify a virus's host
Dark matter (viral)	viral sequences with no close match in current reference databases
Decoy / negative control	a known non-target or blank sample used to detect contamination and false positives
Dereplication	collapsing near-identical sequences into representative clusters (e.g. vOTUs)

Term	Meaning
Hallmark gene	gene strongly diagnostic of viral identity (e.g. capsid, terminase, portal)
Host prediction	computational inference of the likely host of a viral sequence
ICTV taxonomy	the official virus taxonomy framework maintained by the ICTV
Integrase	enzyme that integrates a temperate phage genome into the host chromosome and excises it again during induction
Lysogeny	temperate lifestyle in which a phage genome persists as a prophage
Lytic cycle	infection in which a phage replicates, assembles, and lyses the host to release progeny
MAG (metagenome-assembled genome)	a genome reconstructed by binning contigs from a metagenome
N50	length at which 50% of assembled bases lie in contigs of that length or longer
Provirus / prophage	a viral genome integrated into, or persisting alongside, a host genome
RdRp (RNA-dependent RNA polymerase)	enzyme that replicates RNA virus genomes; a key marker for RNA viruses
Realm	the highest rank in ICTV virus taxonomy (e.g. *Duplodnaviria*)
Terminase	phage enzyme that packages DNA into the capsid; a common hallmark gene
TPM (transcripts per million)	length- and depth-normalized abundance value for cross-sample comparison
Viral metagenomics	metagenomic analysis focused on recovering viral sequences
Virome	the collection of viruses in a sample or ecosystem
Viromics	the sequencing-based study of viral communities
VirSorter2 / geNomad	tools that flag candidate viral sequences in assemblies
vOTU (viral operational taxonomic unit)	a species-level viral cluster, typically defined at 95% ANI over 85% alignment fraction (MIUViG)
Baltimore group	one of the seven classes (I–VII) in the Baltimore system
att site	attachment site where a prophage integrates into or excises from a host genome
Enrichment (VLP)	wet-lab step (filtration, nuclease treatment) that concentrates virus-like particles before sequencing

E Figure and Image Generation Guide

This book uses two kinds of visuals, and this appendix explains how to work with both.

E.1 1. Result figures (data plots)

The plots in Chapter 5 are **real, code-generated figures**. They are produced from the small teaching tables in `data/example/`, so you can reproduce every one without running the heavy pipeline.

```
# From the book root. Figures are written to images/figures/ (Python)
# or visualization/figures/ (R).
python scripts/make_figures.py
Rscript  scripts/make_figures.R
```

To make publication figures from **your own** data, point the scripts at your Phase 4 output tables instead of `data/example/`:

```
python scripts/make_figures.py  my_results/  my_figures/
Rscript  scripts/make_figures.R my_results/  my_figures/
```

Figure file	Shows	Input table
`fig-checkv-quality.png`	CheckV quality tiers	`checkv_quality_summary.tsv`
`fig-contig-length.png`	viral contig lengths	`checkv_quality_summary.tsv`
`fig-top-abundance.png`	mean abundance per vOTU (TPM)	`votu_abundance.tsv`
`fig-abundance-heatmap.png`	vOTU × sample abundance heatmap	`votu_abundance.tsv`
`fig-taxonomy.png`	vOTUs by family	`taxonomy.tsv`
`fig-function.png`	COG functional categories	`functional_summary.tsv`
`fig-host.png`	predicted host phyla	`host_prediction.tsv`
`fig-alpha-diversity.png`	observed richness per sample	`votu_abundance.tsv`

E.2 2. Concept infographics (AI-generated)

Each chapter opens or closes with a **concept infographic**. These ship as branded placeholder images so the book renders cleanly. To finalize one:

1. Open the chapter's collapsible **" Generate this figure"** note and copy its prompt.
2. Generate the image in your preferred tool (for example an image model or a vector editor).
3. Save it over the placeholder at the **same path and filename** listed below. No other edits are needed — the book already references it.

> **Tip**
>
> Keep every concept image at **16:9**, on a **white background**, in the **Codanics palette** (teal `#008b8b`, navy `#05043b`), as a clean flat vector infographic with readable labels. Consistency across chapters is what makes the book look designed rather than assembled.

Chapter	Image file	Concept
1. Fundamentals	`images/ch01-viromics-ecosystem.png`	a virus within a microbial ecosystem feeding a bioinformatics pipeline
2. Experimental design	`images/ch02-experimental-design.png`	from research question to sequencing design
3. Linux setup	`images/ch03-workstation-setup.png`	a reproducible conda + database workstation
4. Complete pipeline	`images/ch04-pipeline-overview.png`	the full assembly-based pipeline, FASTQ to report
5. Visualization	`images/ch05-figure-panels.png`	a multi-panel publication figure
6. Mini project	`images/ch06-wastewater-miniproject.png`	recovering viruses from a public metagenome
7. Practice	`images/ch07-study-map.png`	a skills-consolidation map of the workflow
8. Project ideas	`images/ch08-project-ideas.png`	six research directions in viromics

The full, ready-to-paste prompt for each image lives in that chapter's **" Generate this figure"** note.

E.3 3. Diagrams (Mermaid)

Flowcharts and decision trees in this book are written as Mermaid code blocks (```` ```{mermaid} ````). They render automatically in the HTML and PDF editions, so you can edit them directly in the chapter text — no image files to manage.