Vcflib and tools for processing the VCF variant call format

Erik Garrison¹, Zev N. Kronenberg², Eric T. Dawson³, Brent S. Pedersen⁴, and Pjotr Prins^{1,*}

¹ Dept. Genetics, Genomics and Informatics, University of Tennessee Health Science Center, Memphis, TN, USA

² Pacific Biosciences, San Diego, CA, USA

³ NVIDIA Corporation, Santa Clara, CA, USA

⁴ Center for Molecular Medicine, University Medical Center, Utrecht, The Netherlands

* jprins@uthsc.edu

Abstract

Since its introduction in 2011 the variant call format (VCF) has been widely adopted for processing DNA and RNA variants in practically all population studies — as well as in somatic and germline mutation studies. VCF can present single nucleotide variants, multi-nucleotide variants, insertions and deletions, and simple structural variants called against a reference genome. Here we present over 125 useful and much used free and open source software tools and libraries, part of **vcflib** tools and **bio-vcf**. We also highlight **cyvcf2**, **hts-nim** and **slivar** tools. Application is typically in the comparison, filtering, normalisation, smoothing, annotation, statistics, visualisation and exporting of variants. Our tools run daily and invisibly in pipelines and countless shell scripts. Our tools are part of a wider bioinformatics ecosystem and we consider it very important to make these tools available as free and open source software to all bioinformaticians so they can be deployed through software distributions, such as Debian, GNU Guix and Bioconda. **vcflib**, for example, was installed over 40,000 times and **bio-vcf** was installed over 15,000 times through Bioconda by December 2020. We shortly discuss the design of VCF, lessons learnt, and how we can address more complex variation that can not easily be represented by the VCF format. All source code is published under free and open source software licenses and can be downloaded and installed from <https://github.com/vcflib>.

Keywords: VCF, variant calling, SNP, MNP, INDEL, SV, GFA, C++, Ruby, Python, JSON

Author summary

Most bioinformatics workflows deal with DNA/RNA variations that are typically represented in the variant call format (VCF) — a file format that describes mutations (SNP and MNP), insertions and deletions (INDEL) against a reference genome. Here we present a wide range of free and open source software tools that are used in biomedical sequencing workflows around the world today.

Introduction

From its introduction in 2011 the VCF variant call format has become pervasive in bioinformatics sequencing workflows [1–3]. VCF is one of the important file formats in

bioinformatics workflows because of its critical role in describing variants that come out of sequencing of DNA and RNA. VCF can describe single- and multi- nucleotide polymorphisms (SNP & MNP), insertions and deletions (INDEL), and simple structural variants (SV) against a reference genome [1]. Practically all important variant callers, such as GATK [4] and freebayes [5], produce files in the VCF format. The VCF file format is used in population studies as well as somatic mutation and germline mutation studies. In this paper we discuss the tools we wrote to process VCF and we shortly discuss strengths and shortcomings of the VCF format. We discuss how we can improve future variant calling in its contribution to population genetics.

An important part of the success of VCF that it is a relatively simple and flexible standard that is easy to read, understand and parse. This ‘feature’ has resulted in wide adoption by bioinformatics software developers. VCF typically scales well in bioinformatics workflows because files can be indexed [6], compressed [1, 7, 8] and trivially parallelized in workflows by splitting files and processing them independently, e.g. [5].

Here we present and discuss important tools and libraries for processing VCF in sequencing workflows: `vcflib`, `bio-vcf`, `cyvcf2`, `slivar` and `hts-nim`. These tools were created by the authors for the demands of large VCF file processing and data analysis following the Unix philosophy of small utilities as explained in the ‘small tools manifesto’ [9]. Development of these tools was often driven by the need to transform VCF into other formats, to digest information, to address quality control, and to compute statistics. The `vcflib` toolkit contains both a library and collection of executable programs for transforming VCF files consisting of over 30,000 lines of source code written in the C++ programming language for performance. `vcflib` also comes with a toolkit for population genetics: the Genotype Phenotype Association Toolkit (GPAT). Even though `vcflib` was not previously published, Google scholar shows 132 citations for 2020 alone — pointing to the github source code repository. Next to `vcflib`, we present `bio-vcf`, a parallelized domain-specific language (DSL) for convenient querying and transforming VCF; and we discuss `cyvcf2` [10], `slivar` [11], and `hts-nim` [12] as useful tools and libraries for VCF processing.

Tools and libraries

2.1 `vcflib` C++ tools and libraries

At its core, `vcflib` provides C++ tools and a library application programmers interface (API) to plain text and compressed VCF files. A collection of 83 command line utilities is provided, as well as 44 scripts (Table 1 lists a selection). Most of these tools are designed to be strung together: piping the output of one program into the next, thereby preventing the creation of intermediate files, parallelize processing, and reducing the number IO operations. For example, the `vcflib` `vcfjoincalls` script includes the following pipeline (where `vt` is a variant normalization tool [13]; see Table 1 for the other individual steps by `vcflib`):

```

1 vcfintersect -r $ref \
2     -u <(zcat $base_vcf \
3         | vcfsort -k - \
4         | vt normalize -r $ref -q - 2>/dev/null ) \
5     $aux_vcf \
6     | vcfaddinfo - $aux_vcf \
7     | vcfstreamsort -w 10000000000 \
8     | vcfuniq \
9     | vcfgeno2haplo -r $ref -w 0 \
10    | vcffixup - \
11    | grep -v "^#"

```

Such transformations and statistical functions provided in the toolkit were written for the requirements of projects such as the The 1,000 Genomes Project [14] and NIST's genome in a bottle [15].

The VCF format is a textual file format: each line describes a variant, i.e., a single nucleotide variant (SNV), an insertion, a deletion or a structural variant with rich annotation [1]. In a VCF line, fields are separated by the TAB character. Fields for chromosome, position, the reference sequence, the ALT alleles, and fields for quality, filter, INFO, FORMAT and calls for multiple samples are expected (see Fig. 1). To split fields, for example for 'ALT T,CT' another separator is used; in this case a comma. VCF makes use of many separators by splitting fields into subfields, subsubfields and so on: effectively projecting a 'tree' datastructure onto a single line. The advantage is that it is easy to view a VCF file and it is almost trivial to write a basic VCF parser and it is easy to add information to VCF, sometimes leading to unwieldy nested annotations. An evolving VCF 'standard' is tracked by the samtools/htslib project [17] and later amendments are particularly focused on more complex structural arrangements of DNA/RNA with ALT fields taking somewhat *ad hoc* creative forms, such as 'A[3:67656]' combined with an INFO field containing 'SVTYPE=BND' meaning that starting at reference position on a different chromosome, an ALT A nucleotide is followed by the sequence starting at chromosome 3 and position 67656. These SV annotations do away with some of the original simplicity of VCF. There are many such extensions introduced since the first publication of the VCF 'standard' that are used by specific SV tools and largely ignored by most VCF processing tools.

The `vcflib` API describes a class `vcflib::VariantCallFile` to manage the reading of VCF files, and `vcflib::Variant` to describe the information contained in a single VCF record. The API provides iterators that are used in every included tool. For every record the tree-type hierarchy (Fig. 1) of information can be navigated in the record through interfaces to the fixed fields (CHROM, POS, ID, QUAL, FILTER, INFO) and sample-related fields (FORMAT, and samples). `vcflib` implements functions for accessing and modifying data in these fields; interpreting the alleles and genotypes in record; filtering sites, alleles, and genotypes via a domain-specific filtering boolean language; and reading and writing VCF streams.

In addition to the tools listed in Table 1, `vcflib` includes tools for genotype detection. For example, `abba-baba` calculates the tree pattern for four individuals with an ancestral reference. `vcflib` includes a wide range of tools for transformation, e.g. `vcf2dag` modifies the VCF file so that homozygous regions are included as 'REF/.' calls. For each REF and ALT allele we can assign an index. These steps enable use of the VCF as a partially ordered directed acyclic graph (DAG). `vcfannotate` will intersect the records in the VCF file with targets provided in a BED file and `vcfannotategenotypes` annotates genotypes in the first file with genotypes in the second. `vcfclassify` generates a new VCF where each variant is tagged by allele class: SNP, Ts/Tv, INDEL, and MNP. `vcfglxgt` sets genotypes using the maximum genotype

(a) VCF record

```
1 #CHROM POS ID REF ALT QUAL FILTER INFO FORMAT Original s1t1 s2t1
2 1 10257 . A T,CT 77.69 . AC=1;AF=0.071;AN=14;BaseQRankSum=-0.066;DP=1518;
3 DelS=0.00;FS=24.214;HaplotypeScore=226.5209;MLEAC=1;MLEAF=0.071;
4 MQ=25.00;MQ0=329;MQRankSum=-1.744;QD=0.37;ReadPosRankSum=1.551
5 GT:AD:DP:GQ:PL 0/0:151,8:159:99:0,195,2282
6 0/0:219,22:242:99:0,197,2445 0/0:227,22:249:90:0,90,2339
7 0/0:226,22:249:99:0,159,2695 0/0:166,18:186:99:0,182,1989
8 0/1:185,27:212:99:111,0,2387 0/0:201,15:218:24:0,24,1972
```

(b) JSON representation of VCF record

```
1 {
2   "CHR": "1",
3   "POS": 10257,
4   "REF": "A",
5   "ALT": [
6     "T",
7     "CT"
8   ],
9   "QUAL": 77.69,
10  "DP": 1518,
11  "AF": 0.071,
12  "AN": 14,
13  "MQ": 25,
14  "QD": 0.37,
15  "BaseQRankSum": -0.066,
16  "HaplotypeScore": 226.5209,
17  "samples": {
18    "Original": {
19      "GT": "0/0",
20      "AD": [
21        151,
22        8
23      ],
24      "DP": 159
25    },
26    "s1t1": {
27      "GT": "0/0",
28      "AD": [
29        219,
30        22
31      ],
32      "DP": 242
33    },
34    "s2t1": {
35      "GT": "0/0",
36      "AD": [
37        227,
```

```
38        22
39      ],
40      "DP": 249
41    },
42    "s3t1": {
43      "GT": "0/0",
44      "AD": [
45        226,
46        22
47      ],
48      "DP": 249
49    },
50    "s1t2": {
51      "GT": "0/0",
52      "AD": [
53        166,
54        18
55      ],
56      "DP": 186
57    },
58    "s2t2": {
59      "GT": "0/1",
60      "AD": [
61        185,
62        27
63      ],
64      "DP": 212
65    },
66    "s3t2": {
67      "GT": "0/0",
68      "AD": [
69        201,
70        15
71      ],
72      "DP": 218
73    }
74  }
75 }
```

(c) VCF to JSON transformation

```
1 cat gatk_exome.vcf | bio-vcf --template vcf2json.erb --json
```

Fig 1. Example of the VCF format and a VCF transformation to Javascript Object Notation (JSON) using bio-vcf — (a) the line-based VCF record makes use of separators to split tab-delimited fields into subfields. Subfields are split with characters ,=; / and so on. This splitting effectively projects a ‘tree-like’ datastructure that can also be represented as (b) a JSON record. JSON is used as a common data exchange format for databases and web-services. This example was generated with (c) the bio-vcf tool using a template [16]. bio-vcf transform data to any textual format, including RDF, HTML, XML etc. See also the bio-vcf section

Table 1. A selection of VCF processing tools included with **vcflib** (a full list of over 125 tools with descriptions and options can be found in the online **vcflib** documentation)

Name	Description
Tools for filtering	
vcfaddinfo	add info fields from a second VCF file for records missing in the first file.
vcffixup	insert AC and NS using sample genotypes
vcffilter	filter on fields, e.g. $DP > 10$
vcfuniq	filter out duplicate entries
vcfuniqalleles	filter unique alleles only
Tools for transformation	
vcfintersect	set operations — intersect, union, complement
vcfstremsort	sort
vcfoverlay	merge files by overlaying
vcfcombine	combine samples on identical sites
vcffixup	update fields
vcfannotate	annotate records from BED file
vcfflatten	flatten multi-allelic sites with common ALT genotype
vcfgeno2haplo	transform phased alleles into haplotypes
vcfsampledif	compare VCF files and add annotations to INFO
vcfprimers	design primers
vcf2tsv	convert to tab separated table
vcf2fasta	convert to FASTA
vcf2bed	convert to BED
vcf2sqlite	convert to SQLite
smoother	averages a set of scores over a sliding genomic window
Tools for metrics	
vcfdistance	compute distance between positions and add field
vcfentropy	annotates and add field for sequence entropy for a window
Tools for genotyping	
vcfallelicprimitives	split records if multiple allelic primitives (gaps or mismatches) are specified in a single VCF record
genotypeSummary	summarizes genotype counts for bi-allelic SNVs and INDEL
hapLRT	likelihood ratio test for haplotype lengths
Tools for statistics	
iHS	integrated ratio of haplotype decay between reference and non-reference allele
vFst	compute vFst as a measure of CNV stratification
vcfroc	compute a pseudo-ROC curve
meltEHH	plot extended haplotype homozygosity (EHH) curves
plotHaps	provide output for haplotype plots
popStat	population genetic statistics for each SNP
vcfrandomsample	random sampling
Tools for validation	
vcfcheck	check integrity and identity against reference genome

likelihood for each sample. **vcfinfo summarize** and **vcfsample2info** edit annotations given in the per-sample fields and adds the mean, median, min, or max to the site-level INFO. **vcfleftalign** Left-align INDELs and complex variants in the input using a pairwise REF/ALT alignment followed by a heuristic, iterative left realignment

Fig 2. Smoothed pFst ($-\log_{10}$) statistic with color coded number of variants in a window — as computed by `vcflib`'s `pFst` and `smoother` tools [18].

process that shifts INDEL representations to their absolute leftmost (5') extent.

`vcflib` includes tools for phenotype association based on VCF files. We have developed a flexible and robust genotype-phenotype software library designed specifically for large and noisy NGS datasets. Wright's F-statistics, and especially **Fst**, provide important insights into the evolutionary processes that influence the structure of genetic variation within and among populations, and they are among the most widely used descriptive statistics in population and evolutionary genetics (Fig 2). **Fst** is defined as the correlation between randomly sampled gametes relative to the total drawn from the same subpopulation [19]. **vcFst** is Weir & Cockerham's **Fst** for two populations [20]. **pFst** is a probabilistic approach for detecting differences in allele frequencies between two populations and **bFst** is a Bayesian approach. **bFst** accounts for genotype uncertainty in the model using genotype likelihoods. For a more detailed description see [21]. The likelihood function has been modified to use genotype likelihoods provided by variant callers. There are five free parameters estimated in the model: each subpopulation's allele frequency and **Fis** (fixation index, within each subpopulation), a free parameter for the total population's allele frequency, and **Fst**. **pVst** calculates **Vst** to test the difference in copy numbers at each SV between two groups: $Vst = (Vt - Vs)/Vt$, where **Vt** is the overall variance of copy number and **Vs** the average variance within populations. **sequenceDiversity** calculates two popular metrics of haplotype diversity: **Pi** and extended haplotype homozygosity (**eHH**). **Pi** is calculated using the Nei and Li formulation [22]. **eHH** is a convenient way to think about haplotype diversity. When **eHH**=0 all haplotypes in the window are unique and when **eHH**=1 all haplotypes in the window are identical. The `vcfremap` tool attempts to realign, for each alternate allele, against the reference genome with a lowered gap open penalty and adjusts the CIGAR and REF/ALT alleles accordingly. These traditional and novel population genetic methods are implemented in the Genotype Phenotype Association Toolkit (GPAT++), part of the `vcflib` API. For example, `permuteGPAT++`, adds empirical p-values to a GPAT++ score. And `vcfld` computes linkage disequilibrium (LD). GPAT++ includes basic population stats (**Af**, **Pi**, **eHH**, **oHet**, **genotypeCounts**) and several flavors of **Fst** and tools for linkage, association testing (genotypic and pooled data), haplotype methods (`hapLrt`), smoothing, permutation, and plotting.

`vcflib` includes tools for genotype statistics. `vcfgenosummarize` adds summary statistics to each record summarizing qualities reported in called genotypes. It uses **RO** (reference observation count), **QR** (quality sum reference observations), **AO** (alternate observation count), **QA** (quality sum alternate observations). The `normalizeHS` is used for **iHS** and **XP-EHH** scores [23].

A full list of over 125 commands and functionality can be found on the website, as well as documentation and examples of application [18].

2.2 Bio-vcf and Slivar flexible command-line DSL filters and transformers

2.2.1 bio-vcf

Compared to `vcflib` with its many dedicated command line tools, `bio-vcf` takes a different approach by providing a single command line tool that uses a domain specific language (DSL) for processing the VCF format. Thanks to a dynamic interpretation of the VCF tree representation (see Fig. 1) all data elements in a VCF header or record can be reached using field names and their sub names. For example the following is a valid select filter:

```
1 s.dp>20 and r.filter !~ /~LowQD/ and r.tumor.bcount[r.alt]>4
```

which selects all variants where the sample depth field `s.db` is larger than 20, where the `FILTER` field of a record `r` does not start with the letters *LowQD* (note it uses a Perl/Ruby-style regular expression or regex [24]), and where the tumor bcount of the *ALT* allele is larger than 4. The letter ‘r’ represents a record or line in a VCF file and the letter ‘s’ stands for each sample in a record.

The naming of variables, such as `s.dp` and `r.tumor.bcount`, is inferred from the VCF file itself, so if a VCF has different naming conventions they are picked up automatically.

`bio-vcf` typically reads from the terminal STDIN and writes to STDOUT. The following full command line invocation reads VCF files and filters for chromosomes 1–9 where the quality (`r.qual`) is larger than 50. It also checks for non-empty samples where the sample read depth is larger than 20. For each selected record with `--eval` it outputs a BED record (the default output is the VCF record itself, useful for filtering):

```
1 bio-vcf --filter 'r.chrom.to_i>0 and r.chrom.to_i<=9 and r.qual>50' \
2 --sfilter '!s.empty? and s.dp>20' --eval '[r.chrom,r.pos,r.pos+1]'
```

For comparisons and for output, fields can be converted to integers, floats and strings with `to_i`, `to_f` and `to_s` respectively. Note that these are Ruby functions and, in fact, all such Ruby functionality is available in `bio-vcf` statements. For extreme flexibility `bio-vcf` even supports *lambdas* which makes for very powerful queries and transformations. For example, to output the count of valid genotype fields in samples one could use

```
1 bio-vcf --eval 'r.samples.count {|s| s.gt!="./."}'
```

where `count` is a function that invokes the *lambda* `s.gt!="./."`, i.e., where the genotype `gt` of sample `s` is not equal to “./.”. Sample ‘s’ is passed as a parameter.

Because of the flexibility of `bio-vcf` almost all imaginable data queries can be executed. `bio-vcf` was implemented for processing large VCF files and is fast because it is designed make use of multi-core processors (using Linux parallelized copy-on-write, i.e. a technique where RAM is shared between processes). `bio-vcf` is also ‘lazy’ which means that it only parses fields when they are used. For example, in the above query, only the sample *GT* field is unpacked and parsed to get a result. All other data in the record is ignored by the query and not evaluated. This contrasts largely with most VCF parsers in use today.

Finally, `bio-vcf` comes with a full parser and lexer that can tokenize the VCF file header and transform that in some other format. For example, the command


```
1 bio-vcf --eval-once 'header.meta' --json < gatk_exome.vcf
```

will turn the metadata information passed by GATK [4] into a JSON document. To get a full JSON document of the VCF file use a template that looks like

```
1 =HEADER
2 {
3   "HEADER": {
4     "options": <%= options.to_h.to_json %>,
5     "files":   <%= ARGV %>,
6   },
7   "BODY": [
8 =BODY
9     {
10      "seq:chr": "<%= rec.chrom %>",
11      "seq:pos": "<%= rec.pos %>",
12      "seq:ref": "<%= rec.ref %>",
13      "seq:alt": "<%= rec.alt[0] %>",
14      "dp":      "<%= rec.info.dp %>"
15    }
16 =FOOTER
17   ]
18 }
```

and run

```
1 bio-vcf --template vcf2json.erb --json < gatk_exome.vcf
```

The high expressiveness and adaptable parsing makes **bio-vcf** a very powerful tool for searching, filtering and rewriting VCF files. See the **bio-vcf** website for full information on record and sample inclusion/exclusion filters, generators, multicore performance, field computations, statistics, genotype processing, set analysis and templates for user definable output, including templates for output of VCF header information and records for RDF, JSON, LaTeX, HTML and BED formats [16].

2.2.2 Slivar

Similar to **bio-vcf**, **slivar** allows users to specify simple expressions for filtering and annotation [11]. Whereas **bio-vcf** uses Ruby to supply the DSL, **slivar** uses Javascript. **slivar** has built-in pedigree support for the samples so that, for example, a single expression can be applied to every trio (mother, father, child) to identify *de novo* variants:

```
1 slivar expr --ped $pedigree_file --trio "denovo: kid.het && mom.hom_ref
&& dad.hom_ref && kid.GQ > 10 && mom.GQ > 10 && dad.GQ > 10 && INFO.
gnomad_af < 0.001"
```

The expression above checks the genotype pattern along with genotype quality and limits to rare variants by `INFO.gnomad_af < 0.001`.

Expressions on families (including multigenerational) and arbitrary groups are supported so that, for example, expressions can be applied to tumor-normal pairs using **tumor** and **normal** labels.

2.3 VCF programming libraries

VCF programming libraries are mainly useful when direct calls to **vcflib** and **bio-vcf** command line tools proves too limited. The Bio* libraries, e.g., biopython [25], bioperl [26], bioruby [27] and R's CRAN [28], contain VCF parsers that may be useful. But a first point of call may be **vcflib** itself as it is also a C++

programming library and in addition to being an integral part of the `vcflib` tools mentioned here is used by, for example, the `freebayes` variant caller [5].

Of particular interest is the fast `cyvcf2` library that was started in 2016 with `htslib` [17] bindings and is actively maintained by co-author Pedersen today [10]. Similar to `bio-vcf` it presents a DSL-type language that can be used in Python programming. Meanwhile `hts-nim` [12] contains bindings for the Nim programming language with similar syntax and functionality. For example a nim bin counter (part of `hts-nim-tools/vcf-check` [12]) can be written as:

```
1  # nim code to count variants in bins
2  for variant in v.query(chrom):
3      n += 1
4      if variant.info.get("AF", afs) != Status.OK: continue
5      if max(afs) < maf: continue
6      var bin = which_bin(variant.start.int, chunk_size)
7      counts.extend(bin)
8      inc(counts[bin])
```

Nim is a statically typed compiled language that looks very similar to Python and, because it transpiles to C, Nim has a much faster runtime and can link without overheads against C libraries such as `vcflib` and `htslib`.

Finally, another tool of interest, by the same author, is `vcfanno`; written in the Go language and allows annotations of a VCF with any number of INFO fields from any number of VCFs or BED files. `vcfanno` uses a simple conf file to allow the user to specify the source annotation files and fields and how they will be added to the info of the query VCF [29].

Discussion

Ten years is a long time in bioinformatics and the VCF file format is starting to show its age. Not only is the VCF format redundant and bloated with duplication of data, a more important concern is that the VCF format does not accommodate interesting complex genomic variations, such as complex and nested variants, such as superbubbles, ultrabubbles, and cacti [30–32]. An even more important shortcoming of VCF is that it always depends on a single reference genome, resulting in variant calling bias and missing out on variation not represented in the reference [32]. One solution is to work with multiple reference genomes, but comparing VCF files from different reference genomes is challenging — even for different versions of one reference genome.

To address such challenges the authors are actively working on pangenome approaches that store variation in a pangenome graph format, e.g. [32–37]. Pangenomes can incorporate multiple individuals and multiple reference genomes. Pangenomes can cater for very complex structural variation. Pangenomes are also efficient in storing information, including metadata, without redundancy. In effect, pangenomes cater for a ‘lossless’ view of all data at the population level. This largely differs from VCF-type data because, despite mentioned data size, a generated VCF implies a data reduction step — or data loss — that effectively disconnects variants from each other and related features, such as quality metrics. This means that rebuilding the original data from VCF files is virtually impossible. In contrast, with the newer pangenome formats it is possible to rebuild sequences independent of the underlying complexity of features. Having a full view of the data makes downstream analysis, such as population genotyping, more powerful with improved results, e.g. [32].

The VCF file format has become a crucial part of almost all sequencing workflows today. The design and presentation of the VCF file format can set the norm for designing future file formats [2, 3], but we can also learn from its mistakes. In this

`vcflib`, `bio-vcf` and `cyvcf2`, in contrast, show a continuous adoption of code contributions thanks to the original authors encouraging others to take ownership and even release versions of the software. We also recognise the importance of creating small tools that can interact with each other following the Unix philosophy.

For overall adoption of software solutions it is important the tools and documentation get packaged by software distributions, such as Bioconda [43], Debian [44] and GNU Guix [45, 46]. Bioconda downloads are a good estimation of relative popularity because they tend to represent actual installations. `vcflib`, for example, was installed over 40,000 times and `bio-vcf` was installed over 15,000 times through Bioconda by December 2020. `vcflib` is also an integral part of the freebayes variant caller with an additional 110,000 downloads through Bioconda.

Future work: Software development never stands still. With new requirements tools get updated. With the evolving VCF ‘standard’ and associated tooling for pangenomes and reference based approaches we keep updating our tools and libraries. We intend to add more documentation and regression tests. One recent innovation in `vcflib` is the generation of Unix ‘man’ pages from markdown pages and the `--help` output from every individual tool and script that also doubles as regression tests. This documentation is always up to date because when it goes out of sync the embedded tests fail. We think it will also be useful to add tool descriptions in the common workflow language (CWL) format [46–48]. CWL definitions allow easy sharing of tool components between sequencing workflows. The scenario will be to write a CWL tool definition that can be converted to documentation and running tests.

Despite our criticism of VCF, VCF as a file format is likely to remain in use. To replace VCF most existing tools and workflows used in sequencing would have to be rewritten. Pangenome tools, in principle, are capable of producing reference guided VCF files from GFA graph structures. These tools guarantee compatibility with upstream and downstream analysis workflows. We predict that pangenome approaches will play an increasingly important role in sequence analysis and, at the same time, VCF processing tools will remain in sequencing workflows for the foreseeable future.

Acknowledgments

These free software tools are a community effort and benefit from ideas by hundreds of people. We thank all collaborators, supervisors and other contributors. We particularly want to thank software packagers, such as Andreas Tille and Michael R. Crusoe for providing these tools in Debian. We also want to thank Brad Chapman and Torsten Seemann for the Bioconda packages and Efraim Flashner for the GNU Guix packages.

References

1. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, et al. The variant call format and VCFtools. *Bioinformatics*. 2011;27(15):2156–2158. doi:10.1093/bioinformatics/btr330.

2. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, et al. The variant call format and VCFtools. *Bioinformatics*. 2011;27(15):2156–2158. doi:10.1093/bioinformatics/btr330.
3. HTS-Specs: specifications of SAM/BAM and related high-throughput sequencing file formats; 2011 (accessed April 2021). <https://samtools.github.io/hts-specs/>. GitHub Repository.
4. McKenna A, Hanna M, Banks E, Sivachenko A, Cibulskis K, Kernytsky A, et al. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res*. 2010;20(9):1297–1303. doi:10.1101/gr.107524.110.
5. Garrison E, Marth G. Haplotype-Based Variant Detection from Short-Read Sequencing. 2012;arXiv:abs/1207.3907.
6. Li H. Tabix: fast retrieval of sequence features from generic TAB-delimited files. *Bioinformatics*. 2011;27(5):718–719. doi:10.1093/bioinformatics/btq671.
7. Danecek P, Bonfield JK, Liddle J, Marshall J, Ohan V, Pollard MO, et al. Twelve years of SAMtools and BCFtools. *Gigascience*. 2021;10(2). doi:10.1093/gigascience/giab008.
8. Lan D, Tobler R, Souilmi Y, Llamas B. genozip: a fast and efficient compression tool for VCF files. *Bioinformatics*. 2020;36(13):4091–4092. doi:10.1093/bioinformatics/btaa290.
9. Prins P, Strozzi F, Tarasov A, de Ligt J, Githinji G, others. Small tools MANIFESTO for Bioinformatics; 2014. doi: 10.5281/zenodo.11321. Available from: <https://dx.doi.org/10.5281/zenodo.11321>.
10. Pedersen BS, Quinlan AR. cyvcf2: fast, flexible variant analysis with Python. *Bioinformatics*. 2017;33(12):1867–1869. doi:10.1093/bioinformatics/btx057.
11. Pedersen BS, Brown JM, Dashnow H, Wallace AD, Velinder M, Tvrdik T, et al. Effective variant filtering and expected candidate variant yield in studies of rare human disease. *BioRxiv*. 2020;.
12. Pedersen BS, Quinlan AR. hts-nim: scripting high-performance genomic analyses. *Bioinformatics*. 2018;34(19):3387–3389. doi:10.1093/bioinformatics/bty358.
13. Tan A, Abecasis GR, Kang HM. Unified representation of genetic variants. *Bioinformatics*. 2015;31(13):2202–2204. doi:10.1093/bioinformatics/btv112.
14. Auton A, Brooks LD, Durbin RM, Garrison EP, Kang HM, Korbel JO, et al. A global reference for human genetic variation. *Nature*. 2015;526(7571):68–74. doi:10.1038/nature15393.
15. Zook JM, McDaniel J, Parikh H, Heaton H, Irvine SA, Trigg L, et al. Reproducible integration of multiple sequencing datasets to form high-confidence SNP, indel, and reference calls for five human genome reference materials. *bioRxiv*. 2018;doi:10.1101/281006.
16. bio-vcf: smart VCF parser; 2021 (accessed Feb 2021). <https://github.com/vcflib/bio-vcf>. GitHub Repository.

17. Bonfield JK, Marshall J, Danecek P, Li H, Ohan V, Whitwham A, et al. HTSlib: C library for reading/writing high-throughput sequencing data. *Gigascience*. 2021;10(2). doi:10.1093/gigascience/giab007.
18. vcflib for working with VCF files; 2021 (accessed Feb 2021). <https://github.com/vcflib/vcflib>. GitHub Repository.
19. Holsinger KE, Weir BS. Genetics in geographically structured populations: defining, estimating and interpreting F(ST). *Nat Rev Genet*. 2009;10(9):639–650. doi:10.1038/nrg2611.
20. Cockerham CC, Weir BS. Estimation of gene flow from F-statistics. *Evolution*. 1993;47(3):855–863. doi:10.1111/j.1558-5646.1993.tb01239.x.
21. Holsinger KE, Lewis PO, Dey DK. A Bayesian approach to inferring population structure from dominant markers. *Mol Ecol*. 2002;11(7):1157–1164. doi:10.1046/j.1365-294x.2002.01512.x.
22. Nei M, Li WH. Mathematical model for studying genetic variation in terms of restriction endonucleases. *Proc Natl Acad Sci U S A*. 1979;76(10):5269–5273. doi:10.1073/pnas.76.10.5269.
23. Sabeti PC, Varilly P, Fry B, Lohmueller J, Hostetter E, Cotsapas C, et al. Genome-wide detection and characterization of positive selection in human populations. *Nature*. 2007;449(7164):913–918. doi:10.1038/nature06250.
24. Friedl JEF, Oram A. Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools. In a Nutshell Series. O'Reilly; 1997. Available from: <https://books.google.nl/books?id=qEsPAQAAMAAJ>.
25. Cock PJ, Antao T, Chang JT, Chapman BA, Cox CJ, Dalke A, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*. 2009;25(11):1422–1423.
26. Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, et al. The Bioperl toolkit: Perl modules for the life sciences. *Genome Res*. 2002;12(10):1611–1618.
27. Goto N, Prins P, Nakao M, Bonnal R, Aerts J, Katayama T. BioRuby: bioinformatics software for the Ruby programming language. *Bioinformatics*. 2010;26(20):2617–2619. doi:10.1093/bioinformatics/btq475. Open access.
28. Knaus BJ, Grünwald NJ. VCFR: a package to manipulate and visualize variant call format data in R. *Molecular Ecology Resources*. 2017;17(1):44–53.
29. Pedersen BS, Layer RM, Quinlan AR. Vcfanno: fast, flexible annotation of genetic variants. *Genome Biol*. 2016;17(1):118. doi:10.1186/s13059-016-0973-5.
30. Paten B, Eizenga JM, Rosen YM, Novak AM, Garrison E, Hickey G. Superbubbles, Ultrabubbles, and Cacti. *Journal of Computational Biology*. 2018;25(7):649–663. doi:10.1089/cmb.2017.0251.
31. Paten B, Novak AM, Eizenga JM, Garrison E. Genome Graphs and the Evolution of Genome Inference. *Genome Research*. 2017;27(5):665–676. doi:10.1101/gr.214155.116.

32. Garrison E, Siren J, Novak AM, Hickey G, Eizenga JM, Dawson ET, et al. Variation Graph Toolkit Improves Read Mapping by Representing Genetic Variation in the Reference. *Nature Biotechnology*. 2018;36(9):875–879. doi:10.1038/nbt.4227.
33. Graphical Fragment Assembly (GFA) Format Specification; 2015 (accessed Jan 2021). <https://github.com/GFA-spec/GFA-spec>. GitHub Repository.
34. vgtools for Working with Genome Variation Graphs; 2014 (accessed Jan 2021). <https://github.com/vgteam/>. GitHub Repository.
35. Pangenome Tools; 2020 (accessed Jan 2021). <https://github.com/pangenome/>. GitHub Repository.
36. Pangenome Tools; 2020 (accessed Jan 2021). <https://pangenome.github.io/>. GitHub Repository.
37. pgg: pangenome graph builder; 2020 (accessed Jan 2021). <https://github.com/pangenome/pgg>. GitHub Repository.
38. Lin MF, Bai X, Salerno WJ, Reid JG. Sparse Project VCF: efficient encoding of population genotype matrices. *bioRxiv*. 2020;doi:10.1101/611954.
39. GFF-Spec: Generic Feature Format Version 3 (GFF3); 2016 (accessed April 2021). GFF3 Specification. GitHub Repository.
40. Wilkinson MD, Dumontier M, Aalbersberg IJ, Appleton Ge, Axton M, Baak A, et al. The FAIR Guiding Principles for scientific data management and stewardship;3:160018. doi:10.1038/sdata.2016.18.
41. Holland RC, Down TA, Pocock M, Prlic A, Huen D, James K, et al. BioJava: an open-source framework for bioinformatics. *Bioinformatics*. 2008;24(18):2096–2097.
42. Prins P, de Ligt J, Tarasov A, Jansen RC, Cuppen E, Bourne PE. Toward effective software solutions for big biology. *Nat Biotechnol*. 2015;33(7):686–687. doi:10.1038/nbt.3240.
43. Grüning B and Dale R and Sjödin A and Rowe J and Chapman B and Tomkins-Tinch CH and Valieris R and Köster J. Bioconda: A sustainable and comprehensive software distribution for the life sciences. *bioRxiv*. 2017;doi:10.1101/207092.
44. Debian Linux Software Distribution; 1993 (accessed April 2021). <https://debian.org/>. Online Webpage.
45. Bavier E, Courtès L, Garlick P, Prins P, Wurm R. Guix-HPC Activity Report 2017–2018. Inria Bordeaux Sud-Ouest ; Max Delbrück Center for Molecular Medicine ; Cray, Inc. ; Tourbillon Technology; 2019. Available from: <https://hal.inria.fr/hal-02056461>.
46. Prins P. Creating a reproducible workflow with CWL; 2019. Online. GNU GUIX HPC BLOG.

47. Amstutz P and Crusoe MR and Tijanić N and Chapman B and Chilton J and Heuer M and Kartashov A and Kern J and Leehr D and Ménager H and Nedeljkovich M and Scales M and Soiland-Reyes S and Stojanovic L . Common Workflow Language, v1.0; 2016. Available from: <http://dx.doi.org/10.6084/m9.figshare.3115156.v2>.
48. Strozzi F, Janssen R, Wurmus R, Crusoe MR, Githinji G, Di Tommaso P, et al. Scalable Workflows and Reproducible Data Analysis for Genomics. *Methods Mol Biol.* 2019;1910:723–745. doi:10.1007/978-1-4939-9074-0_24.