

# VcfR: a package to manipulate and visualize VCF data in R

Brian J. Knaus<sup>1</sup> and Niklaus J. Grünwald<sup>1,2</sup>

<sup>1</sup>Horticultural Crops Research Unit, USDA-ARS, Corvallis, 97330, USA

<sup>2</sup>Department of Botany and Plant Pathology, Oregon State University, Corvallis, OR, 97331, USA

**Keywords:** data visualization; high throughput sequencing; quality control; VCF format

**Corresponding author:**

Niklaus J. Grünwald

Horticultural Crops Research Unit, USDA-ARS,  
Corvallis, 97330, USA

Fax: (541) 738-4025

email: Nik.Grunwald@ars.usda.gov

**Running title:** vcfR: manipulate and visualize VCF data

## <sup>1</sup> Abstract

<sup>2</sup> Software to call single nucleotide polymorphisms or related genetic variants has converged  
<sup>3</sup> on the variant call format (VCF) as the output format of choice. This has created a need  
<sup>4</sup> for tools to work with VCF files. While an increasing number of software exists to read  
<sup>5</sup> VCF data, many only extract the genotypes without including the data associated with each  
<sup>6</sup> genotype that describes its quality. We created the R package vcfR to address this issue. We  
<sup>7</sup> developed a VCF file exploration tool implemented in the R language because R provides an  
<sup>8</sup> interactive experience and an environment that is commonly used for genetic data analysis.  
<sup>9</sup> Functions to read and write VCF files into R as well as functions to extract portions of the  
<sup>10</sup> data and to plot summary statistics of the data are implemented. VcfR further provides the

11 ability to visualize how various parameterizations of the data affect the results. Additional  
12 tools are included to integrate sequence (FASTA) and annotation data (GFF) for visualiza-  
13 tion of genomic regions such as chromosomes. Conversion functions translate data from the  
14 vcfR data structure to formats used by other R genetics packages. Computationally intensive  
15 functions are implemented in C++ to improve performance. Use of these tools is intended  
16 to facilitate VCF data exploration, including intuitive methods for data quality control and  
17 easy export to other R packages for further analysis. VcfR thus provides essential, novel tools  
18 currently not available in R.

19

## 20 Introduction

21 Genetic sequence variations are stored in variant call format (VCF) data files for bioinformatic  
22 and genetic analysis. Bioinformatic tools for calling variants such as SAMtools [16] or the  
23 GATK's haplotype caller [1, 6, 18] have all converged on VCF [5, 23] as an output file format.  
24 VCF files have become the de facto standard for downstream analyses of variant data. Prior  
25 to analysis, genetic variants should be screened and filtered based on quality metrics provided  
26 by the variant callers such as SAMtools or GATK-HC. This is particularly important for non-  
27 model organisms where curated panels of variants are not currently available. Convenient and  
28 interactive tools for manipulating these data are not currently available in the R language, a  
29 popular environment for working with genetic data.

30 A VCF file consists of a text file structured according to the VCF definition [5, 23]. These  
31 files may be provided as text files or as compressed files, most typically in the gzip format.  
32 Each file begins with a meta region where abbreviations used in the data portion of the file  
33 are defined with one definition provided per line. The tabular data portion of the file includes  
34 samples in columns and variants in rows. The first eight columns contain information that

describes each variant. The ninth column contains the FORMAT specification for genotype records. All subsequent columns contain sample information where each column corresponds to a sample and the values within each column consist of genotypes and any optional information specified by the FORMAT column. This definition allows a flexible amount of information to be included with each genotype. Variant data can include a SNP, deletion, insertion, large structural variants and/or complex events. However, because the data are not strictly tabular, it presents a challenge in that it needs to be parsed. The quantity of data contained in these files requires this parsing to be performed in an efficient and reproducible manner.

Among currently existing tools for working with VCF files is a collection of tools called VCFtools [5]. VCFtools provides an extensive set of tools for data filtering and analysis. Because they are command line tools they are ideal for high performance computing environments which lack graphical user environments or are implemented in cloud-based queueing systems lacking interactive visualization. However, some users may prefer a more interactive set of tools. Frequently, a set of quality control criteria are used to filter data with no validation of how these criteria may affect the resulting dataset. A graphical and interactive method to manipulate these files would allow researchers to rapidly determine how choices in filtering may affect the resulting dataset.

We present vcfR, an R package designed to help users manipulate and visualize VCF data. Functions are included that efficiently read VCF data into memory and write it back to disk. A parsing function efficiently extracts matrices of genotypes or their associated information. Plotting functions provide a rapid way to visually assess variant characteristics. Because this software is implemented in R it also provides ready access to the multitude of statistical and graphical tools provided by the R environment. Through efficient parsing and visualization, vcfR provides a tool to rapidly develop hard filters for quality metrics that can be easily tailored to individual projects and experimental designs. Key components of vcfR

61 are implemented in C++ and called from R to minimize time of computation. VcfR is open  
62 source and available on CRAN and GitHub with appropriate documentation.

## 63 Materials and Methods

### 64 The pinfsc50 example data set

65 An example data set for the diploid, oomycete pathogen *Phytophthora infestans* is provided  
66 to demonstrate application of vcfR. We use sequence data from supercontig\_1.50 in the  
67 FASTA format from the published genome [11, 2] as example sequence data. We also provide  
68 an annotation file in the GFF format for supercontig\_1.50. Lastly, we provide the variant  
69 data in VCF format. VCF format data is based on short read data from previously published  
70 sources [4, 2, 17, 29]. This short read data was mapped to the T30-4 reference with bwa-mem  
71 [15], while bam improvement and variant calling was performed according to the GATK's best  
72 practices [1, 6, 18]. Phasing was performed with beagle4 [3]. Because beagle4 removes most of  
73 the diagnostic information output by the GATK's haplotype caller, the phased genotype from  
74 the beagle4 VCF file was appended to the haplotype caller's VCF file (after the unphased  
75 genotype was removed), resulting in the VCF file provided. It is important to note that  
76 we have processed our own data as many publicly available datasets lack the richness of  
77 descriptors provided by variant calling software. For example, data available from the 1,000  
78 genomes project only contains the genotypes and none of the quality metrics (i.e., it is a  
79 production dataset). By providing our own processing it allows for provision of VCF format  
80 files rich with information that can be used to provide instructive examples.

81 Ideally, this example dataset will be somewhat large so that it demonstrates efficient  
82 execution of functions intended to be used on typically larger VCF data sets. However,  
83 packages hosted on CRAN currently generate a 'NOTE' when package size exceeds 5MB.  
84 This arbitrary threshold creates a practical limitation to ensure a data set is small enough

85 to distribute efficiently. In order to balance this need for size we have released the pinfsc50  
86 dataset on CRAN as its own R package with the same name and hope others will find it  
87 useful as well.

88 **Efficient file access**

89 A common bottleneck to data intense projects is reading and writing files from disk to  
90 memory, and back to disk. Essential criteria for VCF file input and output include that  
91 it must be fast, able to read text and gzipped files, and able to handle the non-tabular  
92 VCF format. Furthermore, input ideally should allow subsetting to specific rows (variants)  
93 and columns (samples) of interest. This allows work to be performed on datasets where  
94 the entire file may require more physical memory than is available in a given computational  
95 environment. We created a custom VCF file reading function using Rcpp [9] to execute the  
96 computationally intensive steps in C++. This C++ code allows efficient reading and writing  
97 of VCF files.

98 To provide a comparison among our new function and existing R functions we conducted  
99 a benchmarking test. Evaluation was run on a 3.60GHz Intel® Core™ i7-4790 CPU running  
100 Ubuntu 12.04.5 LTS with a Western Digital® WDC WD10EZEX-75M drive. Our test data  
101 set, pinfsc50, was a gzipped VCF file containing 29 meta lines, 22,031 variants and 27 columns  
102 (18 samples). Functions to read tabular data (*utils::read.table()* and *data.table::fread()*) were  
103 parameterized to skip the non-tabular meta region, providing a slight advantage to these  
104 functions. The function *data.table::fread()* was called as *data.table::fread('zcat filename.gz')*  
105 because it does not currently read in gzipped data. Data input was run twice to determine  
106 if functions implemented some form of caching that may improve a second read time. Input  
107 time was measured with the R function *system.time()* where elapsed time was recorded.

## 108 Manipulation of VCF data

109 Once the VCF data is read into memory, manipulation of the data follows. The data for each  
110 genotype from a VCF file can be seen as a colon delimited string containing the genotype as  
111 well as optional information to characterize each genotype. The FORMAT column specifies  
112 the format of these elements. For example, many variant callers will provide information  
113 on how many times each allele was sequenced, genotype likelihoods and other information  
114 to characterize the quality of each genotype. An example of a FORMAT element and a  
115 genotype element is provided in equation 1:

$$\text{GT:AD:DP:GQ:PL } 0|0:5,0:5:15:0,15,185 \quad (1)$$

116 Here ‘GT’ is the genotype (0|0, homozygous for the reference) and allelic depth (AD) for  
117 the genotype is ‘5,0’ (the reference allele was sequenced five times and the alternate allele  
118 was sequenced zero times). The definition of these abbreviations should be found in the meta  
119 region of the file. As the number of variants and samples increases, the task of parsing this  
120 data increases multiplicatively. This presents a performance challenge to typical R functions.  
121 We included a parser (the function *extract.gt()*) that is implemented using Rcpp [9] to rapidly  
122 extract this information. The result is a matrix of strings or numerics corresponding to one  
123 of the specified FORMAT values. These matrices can then be manipulated as typical R  
124 matrices and be visualized using the base R graphics system [21] or other tools.

## 125 Chromosomal summaries

126 Variants for VCF data can be incorporated with sequence and annotation data to provide  
127 chromosomal perspectives. Within the vcfR framework this process results in the creation of  
128 an object of class chromR. The vcfR function *chromoqc()* can be used to visualize chromR  
129 objects. During the creation and subsequent processing of this object, summary statistics

130 of the variants are provided (heterozygosity, effective size) and sliding window analyses are  
131 performed (nucleotide content, number of annotated positions, variants per window). These  
132 per variant and per window summaries are stored in a tabular format that can be visualized  
133 with the R base graphics system (e.g., *graphics::hist()* and *graphics::plot()*) or saved to a  
134 file. A genomic perspective can then be obtained by concatenating chromosomal summaries.  
135 Integration of variant data with sequence and annotation data provides a novel tool to rapidly  
136 identify genomic regions of interest.

## 137 Results

138 The vcfR package includes novel functions for reading and writing data from VCF files and  
139 for visualizing, manipulating and quality filtering of data.

### 140 File access

141 We wrote code for reading and subsetting VCF data in C++ to improve computational  
142 speed. Results from the file access benchmarking test are presented in table 1. In general,  
143 the typical R functions for data input, such as *utils::read.table()* do not perform well. The  
144 function *utils::read.table()* required 2-3 seconds to read in our modest size data set (18 sam-  
145 ples, 22 thousand variants). More efficient methods of reading tabular data into R include  
146 *read\_table()* in the package *readr* [28] and *fread()* in the package *data.table* [8]. The func-  
147 tion *data.table::fread()* is perhaps the best performing function with the greatest flexibility  
148 in terms of column and row access. It read data into memory in 0.1-0.2 seconds. However,  
149 it lacks the ability to read gzipped files and is designed for reading only tabular data. On  
150 Unix systems, gzipped data may be read in by modifying the call to include *zcat* or *gzcat*.  
151 Windows users may be required to make an uncompressed version of their file to facilitate  
152 use, resulting in redundancy of files, a practice that is unnecessary. These general functions

153 that read in tabular data may provide a path for data input, however, accessing the result-  
154 ing data for manipulation from the resulting table remains an issue. Because each element  
155 read in by these functions that read in tabular data may contain data associated with each  
156 genotype as a colon delimited string they need further processing before attaining a format  
157 for analysis in existing R genetics packages.

158 In addition to the functions already mentioned above, functions currently exist to read  
159 VCF files into R. The package PopGenome includes the function *readVCF()*. This function  
160 reads in VCF files that are both faidx indexed and bgzipped, making this function beyond  
161 the scope of these comparisons. This may present a direction high throughput projects may  
162 take in the future. At the present, it is our experience that gzipped files are more common.  
163 This package also does not appear to include conversion functions to translate data into data  
164 structures used in commonly used R packages such as ape [20], adegenet [12], pegas [19]  
165 and poppr [14, 13]. This function therefore appears applicable to only this package. The  
166 Bioconductor package VariantAnnotation includes a *readVcf()* function. It does not appear  
167 to perform as well as the other options at reading in data as it required 5.7 and 1.2 seconds to  
168 read in data (Table 1). Also, the resulting data structure appears complicated relative to the  
169 other options presented here and appears specific to the Bioconductor project. Users who  
170 are already invested in the Bioconductor data structures may find this to be a viable route.  
171 If the user is instead interested in analyses from packages from outside of Bioconductor (e.g.,  
172 CRAN), this route is less appealing. The package pegas includes the function *read.vcf()*. This  
173 function appears to fulfill all of our criteria. However, this function only reads in the genotypes  
174 from the VCF file and none of the data associated with each genotype. This may be because  
175 the object of class loci, created by this function, does not support associated information.  
176 This means that this function will not provide information for quality filtering, a task crucial  
177 to obtaining high quality data. This function read in our test file in 2.3 and 0.4 seconds.  
178 The first read is comparable to *utils::read.table()*, suggesting relatively poor performance.

179 However, the second read was comparable to *data.table::fread* and *vcfR::read.vcfR*, suggesting  
180 a well performing input function. The difference in input speed suggests that some sort of  
181 caching may be occurring that makes subsequent reads much faster than initial reads for  
182 this function. Our function *vcfR::read.vcfR()* performs comparably to *data.table::fread()* and  
183 *readr::read\_table()* (Table 1) but provides the convenience that it is specifically designed to  
184 handle VCF data whether compressed (gzip) or not. It also provides the ability to select rows  
185 and columns from a VCF file so that partial files can be read in to conserve memory. Lastly,  
186 this package provides conversion functions to translate data from the vcfR object created by  
187 *vcfR::read.vcfR()* into formats supported by other R genetics packages. Most importantly, as  
188 described below in more detail, our VCF handling includes quality metrics available in input  
189 files for subsequent filtering.

## 190 Data parsing and visualization

191 Once VCF data are read into memory they are ready for further processing. However, because  
192 the data are not in a strictly tabular format (see equation 1) further processing is needed to  
193 access the data. Our function *extract.gt()* parses elements from the gt portion of VCF data.  
194 For example, the read depth at each variant (DP) is commonly included in VCF files either  
195 by default or as an option. Invocation of *extract.gt()* on VCF data containing depths results  
196 in a numeric matrix of depths with samples in columns and variants in rows. This data can  
197 then be visualized with existing R tools, such as ggplot2 [27] which was used to summarize  
198 a matrix of depths using violin plots (Figure 1). Similarly, a modified heatmap function  
199 provided in vcfR (*heatmap.bp()*) can be used to summarize the depth matrix (Figure 2). The  
200 steps to achieve these results are illustrated with the code provided below.

```
201 # Load libraries.  
202 library(vcfR)  
203 library(pinfsc50)  
204 library(reshape2)
```

```
205 library(ggplot2)
206
207 # Find and input VCF data.
208 vcf <- system.file("extdata", "pinf_sc50.vcf.gz", package = "pinfsc50")
209 vcf <- vcfR::read.vcfR(vcf, verbose = FALSE)
210
211 # Parse DP from the gt region.
212 dp <- extract.gt(vcf, element="DP", as.numeric = TRUE)
213
214 # Reorganize and render violin plots.
215 dpf <- melt(dp, varnames=c("Index", "Sample"), value.name = "Depth", na.rm=TRUE)
216 dpf <- dpf[ dpf$Depth > 0,]
217 p <- ggplot(dpf, aes(x=Sample, y=Depth)) + geom_violin(fill="#COCOCO", adjust=1.0,
218 scale = "count", trim=TRUE)
219 p <- p + theme_bw()
220 p <- p + ylab("Read Depth (DP)")
221 p <- p + theme(axis.title.x = element_blank(),
222 axis.text.x = element_text(angle = 60, hjust = 1))
223 p <- p + stat_summary(fun.data=mean_sdl, geom="pointrange", color="black")
224 p <- p + scale_y_continuous(trans=scales::log2_trans(), breaks=c(1, 10, 100, 1000))
225 p
226
227 # Plot as heatmap.
228 heatmap_bp(dp[501:1500,])
```

229 This functionality adds data parsing to file input in vcfR thereby creating an easy path  
230 to visualization of VCF data.

## 231 Quality filtering

232 Variant calling software typically requires some form of post-hoc quality filtering. This is  
233 particularly important in non-model systems that contain only a small portion of curated  
234 data [10]. The vcfR function *chromoqc()* was used to generate Figure 3 from a FASTA  
235 sequence file, a GFF annotation file and a VCF variant file. Inspection of Figure 3 illustrates  
236 a number of issues typically observed in raw VCF data. For example, the read depth (DP) is  
237 highly variable. The marginal box and whisker plot for the read depth panel contains 50% of  
238 the data within the boxes of the box and whisker plot. This provides an estimate of depth to

239 expect for base ploid variants. Below this region may be variants of low coverage that may  
240 not have been called accurately. Above this region are variants that may be from repetitive  
241 regions of the genome and may therefore violate ploidy assumptions made by the variant  
242 caller. Mapping quality (MQ) consists primarily of variants with values of 60 as well as a  
243 population of variants of a lower quality. By using the vcfR function *masker()* we can filter  
244 on thresholds for these values. Here we have used a read depth between 350 and 650 and a  
245 mapping quality between 59.5 and 60.5. The result is visualized in Figure 4. We see that  
246 we eliminated most of the variants on the right hand side of the plot. The resulting data set  
247 may now be considered to be of higher stringency. Alternatively, a researcher may want to  
248 use this variant set to use as a training set for another round of variant discovery. Processing  
249 and visualization of these results can occur with a few lines of code:

```
250 # Load libraries
251 library(vcfR)
252 library(pinfsc50)
253
254 # Determine file locations
255 vcf_file <- system.file("extdata", "pinf_sc50.vcf.gz",
256                         package = "pinfsc50")
257 dna_file <- system.file("extdata", "pinf_sc50.fasta",
258                         package = "pinfsc50")
259 gff_file <- system.file("extdata", "pinf_sc50.gff",
260                         package = "pinfsc50")
261
262 # Read data into memory
263 vcf <- read.vcfR(vcf_file)
264 dna <- ape::read.dna(dna_file, format = "fasta")
265 gff <- read.table(gff_file, sep="\t", quote="")
266
267 # Create a chromR plot
268 chrom <- create.chromR(name="Supercontig",
269                         vcf=vcf, seq=dna,
270                         ann=gff)
271
272 # Mask for read depth and mapping quality
273 chrom <- masker(chrom, min_QUAL=0, min_DP=350,
```

```
274     max_DP=650, min_MQ=59.5,  
275     max_MQ=60.5)  
276 chrom <- proc.chromR(chrom)  
277  
278 # Plot.  
279 chromoqc(chrom, dp.alpha=20)
```

## 280 Data export

281 Data can be exported from vcfR into several data formats useful for downstream analysis.  
282 The most straight forward option may be to output the manipulated data as a VCF file.  
283 The function *write.vcf()* takes a vcfR object and writes it to file as a gzipped VCF file. This  
284 allows any software that operates on VCF files to be used for downstream analysis. If the  
285 researcher prefers to remain in the R environment, several other options exist. The function  
286 *vcfR2genind()* can be used to convert modest amounts of VCF data into a genind object, al-  
287 lowing analysis in adegenet [12]. Genind objects may be easily converted to genclone objects  
288 with *poppr:::as.genclone()* for analysis in poppr [14, 13]. Authors of the adegenet package have  
289 more recently created the genlight object specifically for high throughput sequencing appli-  
290 cations. Objects of class genlight can be created from objects of class vcfR using the function  
291 *vcfR2genlight()*. Once an object of class genlight has been created it can be converted to an  
292 object of class snpclone using *poppr:::as.snpclone()*. When sequence information is provided,  
293 the VCF data can be converted into an object of class DNAbin using *vcfR2DNAbin()* for  
294 analysis in ape [20] or pegas [19]. The inclusion of data conversion functions allows VCF data  
295 to be easily converted into data structures used by currently existing R genetics packages  
296 making these existing methodologies available to the analysis of VCF data.

## 297 Discussion

298 The advent of high throughput sequencing has provided researchers with a deluge of data.  
299 As with all data, some of it is of high quality while some of it may not be. The VCF  
300 file format provides a flexible format that authors of variant callers can use to include a  
301 diversity of information to support genotype calls. However, software available to utilize this  
302 information, particularly in the R environment, is currently limited.

303 The R package vcfR is a novel tool for manipulation and visualization of data contained  
304 in VCF files. This package contains functions to efficiently read and write VCF data from  
305 and to files. Functionality is also provided to parse VCF data once loaded into memory. This  
306 creates an entry point for VCF data analysis in the R environment with its associated genetic  
307 analysis packages. For example, functions in vcfR can read in VCF data, extract numeric  
308 values such as read depth or genotype qualities from this data, and the data can then be  
309 visualized using standard R scatter plots or histograms. These data can also be visualized  
310 with custom plots provided by vcfR. This information can be used to determine thresholds  
311 for quality filtering, similar to VCFtools [5], but in a graphical, interactive R environment.  
312 The package also includes functions to convert this information to formats used by existing R  
313 packages specifically designed to work with population genetic data (e.g., ape [20], adegenet  
314 [12], pegas [19] and poppr [14, 13]). Once VCF data is read into memory, typically a single  
315 function is all that is required to translate the data into a data structure supported by these  
316 other packages. This makes the data contained in VCF files available to functions provided  
317 by the vcfR package, R's standard plotting functions as well as methodologies that currently  
318 exist in available R packages.

319 The integration of vcfR with existing methodologies provides researchers with a rapid  
320 path towards research products. Once VCF data have been generated, they can be rapidly  
321 queried for quality metrics to determine quality filtering thresholds. This information can  
322 then be used to filter data within vcfR or the thresholds can be used in server side processes

323 such as VCFtools [5]. Once VCF data has been determined to be of sufficient quality for  
324 downstream use, vcfR provides data export tools for other analytical tools available in R.  
325 This novel functionality allows VCF data to be easily handled from data acquisition, through  
326 quality control and final analysis within the R programming environment. VcfR provides  
327 flexibility to aid researchers explore quality thresholds and other analytical decisions in an  
328 efficient manner to ensure the lowest amount of technical variation and the highest quality  
329 results.

## 330 Acknowledgements

331 We acknowledge Javier F. Tabima (Department of Botany and Plant Pathology, Oregon  
332 State University) for extensive alpha testing that greatly improved the R code. Zhian N.  
333 Kamvar (Department of Botany and Plant Pathology, Oregon State University) provided  
334 valuable discussion on R package development. N.J.G. thanks NESCent for organizing the  
335 2015 Population Genetics in R Hackathon in Durham, NC. This research is supported in part  
336 by US Department of Agriculture (USDA) Agricultural Research Service Grant 5358-22000-  
337 039-00D and USDA National Institute of Food and Agriculture Grant 2011-68004-30154 to  
338 N.J.G..

## 339 Data Accessibility

340 The program and user manual are available on CRAN (<http://cran.r-project.org/package=vcfR>).  
341 The pinfsc50 dataset is available on CRAN (<http://cran.r-project.org/package=pinfsc50>).

## 342 Author Contributions

- 343 BJK conceived of the project, wrote code, wrote the documentation, and wrote the manuscript.
- 344 NJK coordinated the collaborative effort, discussed interpretation, wrote the manuscript and
- 345 obtained funding.

## 346 References

- 347 [1] Geraldine A Auwera, Mauricio O Carneiro, Christopher Hartl, Ryan Poplin, Guillermo  
348 del Angel, Ami Levy-Moonshine, Tadeusz Jordan, Khalid Shakir, David Roazen, Joel  
349 Thibault, et al. From fastq data to high-confidence variant calls: the genome analysis  
350 toolkit best practices pipeline. *Current Protocols in Bioinformatics*, pages 11–10, 2013.
- 351 [2] Broad Institute of Harvard and MIT. *Phytophthora infestans* sequencing project.  
352 <http://www.broadinstitute.org>, 2015. Accessed: 2015-10-2.
- 353 [3] Sharon R Browning and Brian L Browning. Rapid and accurate haplotype phasing and  
354 missing-data inference for whole-genome association studies by use of localized haplotype  
355 clustering. *The American Journal of Human Genetics*, 81(5):1084–1097, 2007.
- 356 [4] David EL Cooke, Liliana M Cano, Sylvain Raffaele, Ruairidh A Bain, Louise R Cooke,  
357 Graham J Etherington, Kenneth L Deahl, Rhys A Farrer, Eleanor M Gilroy, Erica M  
358 Goss, et al. Genome analyses of an aggressive and invasive lineage of the Irish potato  
359 famine pathogen. *PLoS Pathog*, 8(10):e1002940, 2012.
- 360 [5] Petr Danecek, Adam Auton, Goncalo Abecasis, Cornelis A Albers, Eric Banks, Mark A  
361 DePristo, Robert E Handsaker, Gerton Lunter, Gabor T Marth, Stephen T Sherry, et al.  
362 The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158, 2011.
- 363 [6] Mark A DePristo, Eric Banks, Ryan Poplin, Kiran V Garimella, Jared R Maguire,  
364 Christopher Hartl, Anthony A Philippakis, Guillermo Del Angel, Manuel A Rivas, Matt  
365 Hanna, et al. A framework for variation discovery and genotyping using next-generation  
366 DNA sequencing data. *Nature Genetics*, 43(5):491–498, 2011.
- 367 [7] Maureen J Donlin. Using the generic genome browser (gbrowse). *Current Protocols in*  
368 *Bioinformatics*, pages 9–9, 2009.
- 369 [8] M Dowle, A Srinivasan, T Short, S Lianoglou, with contributions from R Saporta,  
370 and E Antontyan. data.table: Extension of data.frame. <https://cran.r-project.org/web/packages/data.table/index.html>, 2015. Accessed: 2015-12-4.
- 371 [9] Dirk Eddelbuettel and Romain Francois. Rcpp: Seamless R and C++ integration.  
372 *Journal of Statistical Software*, 40:1–18, 2011.

- 374 [10] Niklaus J Grünwald, Bruce A McDonald, and Michael G Milgroom. Population genomics  
375 of fungal and oomycete pathogens. *Annual Review of Phytopathology*, page in press, 2016.
- 376 [11] Brian J Haas, Sophien Kamoun, Michael C Zody, Rays HY Jiang, Robert E Hand-  
377 saker, Liliana M Cano, Manfred Grabherr, Chinnappa D Kodira, Sylvain Raffaele, Trudy  
378 Torto-Alalibo, et al. Genome sequence and analysis of the Irish potato famine pathogen  
379 *Phytophthora infestans*. *Nature*, 461(7262):393–398, 2009.
- 380 [12] Thibaut Jombart. adegenet: a R package for the multivariate analysis of genetic markers.  
381 *Bioinformatics*, 24(11):1403–1405, 2008.
- 382 [13] Zhian N Kamvar, Jonah C Brooks, and Niklaus J Grünwald. Novel R tools for analysis of  
383 genome-wide population genetic data with emphasis on clonality. *Frontiers in Genetics*,  
384 6(208), 2015.
- 385 [14] Zhian N Kamvar, Javier F Tabima, and Niklaus J Grünwald. Poppr: an R package for  
386 genetic analysis of populations with clonal, partially clonal, and/or sexual reproduction.  
387 *PeerJ*, 2:e281, 2014.
- 388 [15] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows–  
389 Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.
- 390 [16] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor  
391 Marth, Goncalo Abecasis, Richard Durbin, et al. The sequence alignment/map format  
392 and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.
- 393 [17] Michael D Martin, Enrico Cappellini, Jose A Samaniego, M Lisandra Zepeda, Paula F  
394 Campos, Andaine Seguin-Orlando, Nathan Wales, Ludovic Orlando, Simon YW Ho,  
395 Fred S Dietrich, et al. Reconstructing genome evolution in historic samples of the Irish  
396 potato famine pathogen. *Nature Communications*, 4, 2013.
- 397 [18] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis,  
398 Andrew Kernytsky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, et al.  
399 The genome analysis toolkit: a mapreduce framework for analyzing next-generation  
400 DNA sequencing data. *Genome Research*, 20(9):1297–1303, 2010.
- 401 [19] Emmanuel Paradis. pegas: an R package for population genetics with an integrated–  
402 modular approach. *Bioinformatics*, 26(3):419–420, 2010.
- 403 [20] Emmanuel Paradis, Julien Claude, and Korbinian Strimmer. APE: analyses of phylogenetics  
404 and evolution in R language. *Bioinformatics*, 20(2):289–290, 2004.
- 405 [21] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation  
406 for Statistical Computing, Vienna, Austria, 2015.
- 407 [22] James T Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S  
408 Lander, Gad Getz, and Jill P Mesirov. Integrative genomics viewer. *Nature Biotechnology*,  
409 29(1):24–26, 2011.

- 410 [23] samtools. Hts-specs. <http://samtools.github.io/hts-specs/>. Accessed: 2016-01-27.
- 411 [24] Mitchell E Skinner, Andrew V Uzilov, Lincoln D Stein, Christopher J Mungall, and Ian H  
412 Holmes. Jbrowse: a next-generation genome browser. *Genome Research*, 19(9):1630–  
413 1638, 2009.
- 414 [25] Lincoln D Stein, Christopher Mungall, ShengQiang Shu, Michael Caudy, Marco Man-  
415 gone, Allen Day, Elizabeth Nickerson, Jason E Stajich, Todd W Harris, Adrian Arva,  
416 et al. The generic genome browser: a building block for a model organism system  
417 database. *Genome Research*, 12(10):1599–1610, 2002.
- 418 [26] Helga Thorvaldsdóttir, James T Robinson, and Jill P Mesirov. Integrative genomics  
419 viewer (IGV): high-performance genomics data visualization and exploration. *Briefings  
420 in Bioinformatics*, page bbs017, 2012.
- 421 [27] Hadley Wickham. *ggplot2: elegant graphics for data analysis*. Springer New York, 2009.
- 422 [28] Hadley Wickham and Romain Francois. *readr: Read Tabular Data*, 2015. R package  
423 version 0.2.2.
- 424 [29] Kentaro Yoshida, Verena J Schuenemann, Liliana M Cano, Marina Pais, Bagdevi Mishra,  
425 Rahul Sharma, Chirsta Lanz, Frank N Martin, Sophien Kamoun, Johannes Krause, et al.  
426 The rise and fall of the *Phytophthora infestans* lineage that triggered the Irish potato  
427 famine. *Elife*, 2:e00731, 2013.

428 **Tables**

Table 1: Comparison of R functions that read VCF or VCF-like data into R. The pinfsc50 data used for performance benchmarking consisted of 18 samples and 22,031 variants in a gzipped VCF file.

	utils:: read.table()	readr:: read_table()	data.table:: fread()	pegas:: read.vcf()	VariantAnnotation:: readVcf()	vcfR:: read.vcfR()
Reads VCF	F	F	F	T	T	T
Reads gzip	T	T	F	T	T	T
Conversion functions	F	F	F	T	F	T
Row selection	T	T	T	T	T	T
Column selection	F	F	T	F	T	T
First read (sec)	3.259	0.637	0.257	2.260	5.739	0.419
Second read (sec)	2.413	0.384	0.177	0.387	1.246	0.377

429 **Figures**

Figure 1: Violin plot of read depth (DP) for the 18 samples in the pinfsc50 data set. A numeric matrix was produced from the VCF file with the function `vcfR::extract.gt()`.

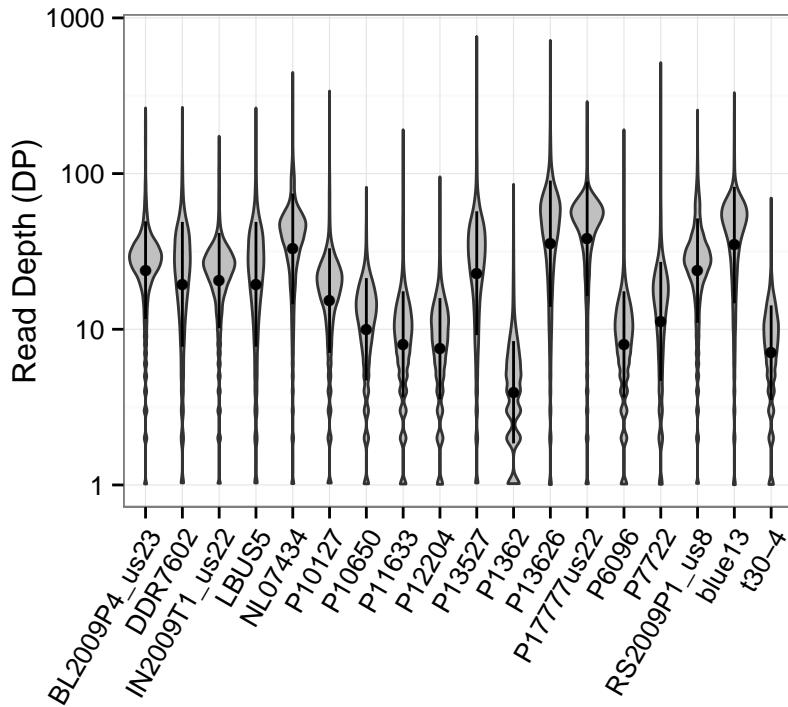


Figure 2: Heatmap of read depth (DP) for the 18 samples in the pinfsc50 data set. Each column is a sample and each row is a variant. The color of each cell corresponds to each variant’s read depth (DP). Cells in white contain missing data. Marginal barplots summarize row and column sums. The matrix was extracted from the VCF data with the function `vcfR::extract.gt()` and the plot generated with `vcfR::heatmap.bp()`.

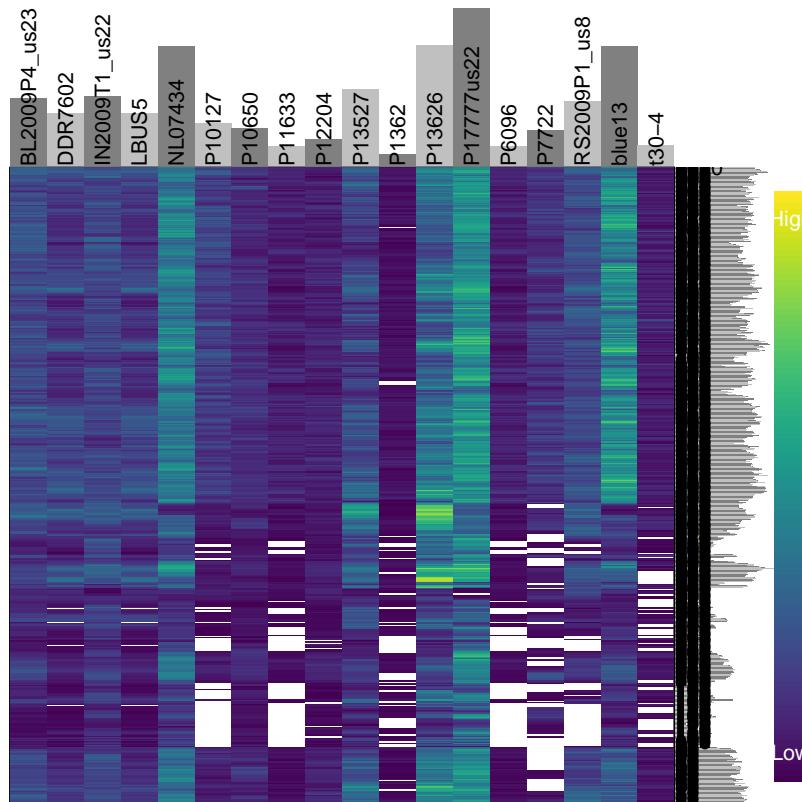


Figure 3: Chromoqc plot showing raw VCF data for one supercontig in the pinfsc50 data set. The lowest panel represents annotations as red rectangles. Above this is a panel where regions of called nucleotides (A, C, G or T) are represented as green rectangles and ambiguous nucleotides (N) are represented as red, narrower, rectangles. Continuing up the plot is a sliding window analyses of GC content and then one of variant incidence. Above these are three dot plots of phred-scaled quality (QUAL), mapping quality (MQ) and read depth (DP).

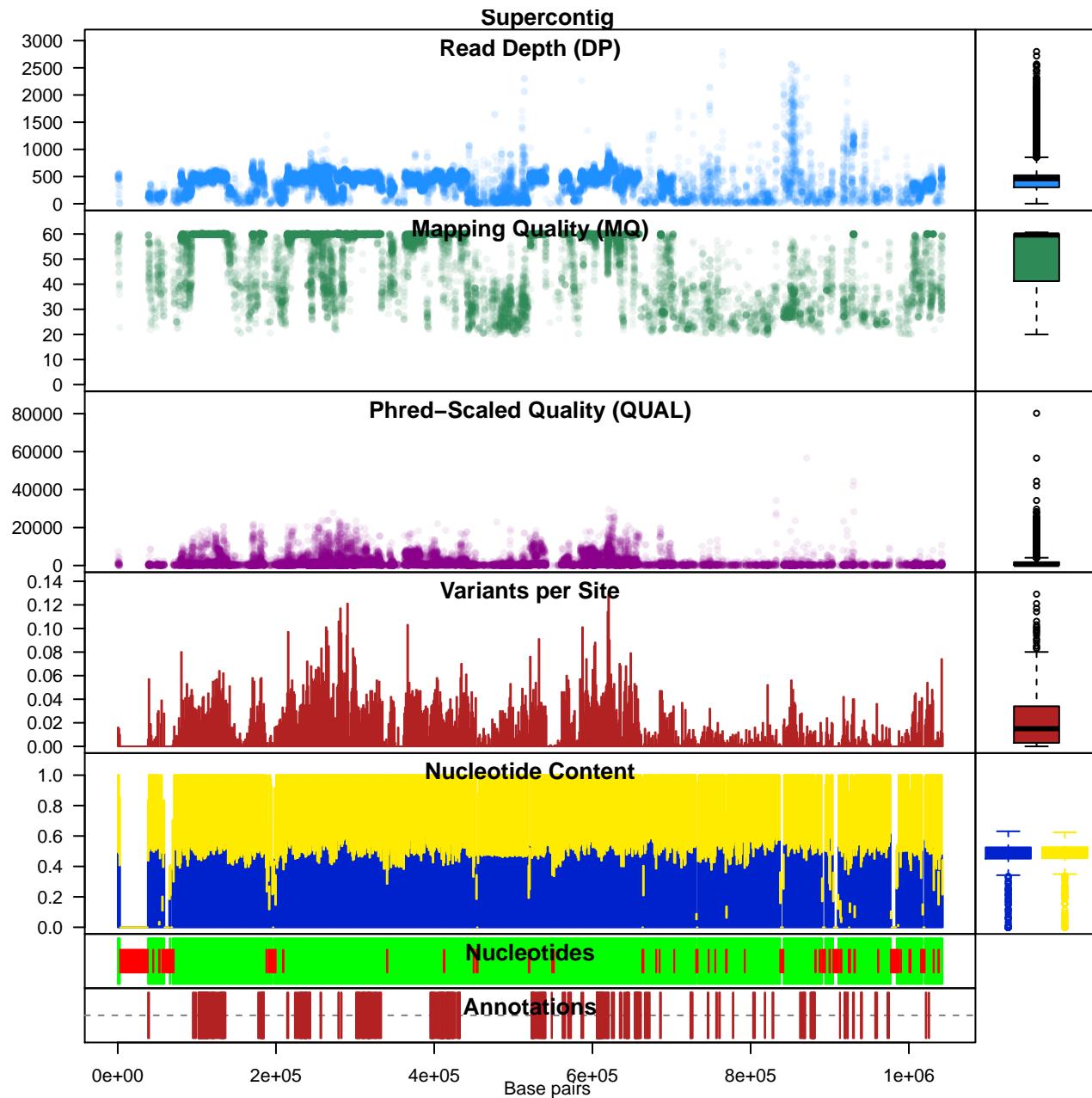


Figure 4: Chromoqc plot showing the one supercontig in the pinfsc50 data set after filtering on read depth (DP) and mapping quality (MQ).

