



viu

**Universidad
Internacional
de Valencia**

GenAgans: una aplicación Shiny interactiva e intuitiva para analizar variantes genéticas humanas

Titulación:
Máster en Bioinformática

Curso académico
2023 - 2024

Alumna: Candelas Galavís,
Blanca

D.N.I: [REDACTED]

Director de TFM: [REDACTED]
[REDACTED]

Convocatoria:
Segunda

Índice

1. Introducción	10
1.1. La bioinformática y la era del <i>Big Data</i>	10
1.2. La genómica y las variantes genéticas en la biomedicina.....	13
1.2.1. Tipos de variantes genéticas	13
1.2.2. Análisis de variantes	16
1.2.3. Relevancia de la predicción de variantes genéticas en la práctica clínica.....	19
1.3. Flujo de trabajo para la identificación y predicción del efecto de variantes genéticas	20
1.4. Desarrollo de flujos de trabajo bioinformáticos.....	22
1.5. Problemática presente y justificación del desarrollo de este trabajo	23
2. Objetivos	24
3. Metodología	25
3.1. Materiales.....	25
3.2. Flujo de trabajo para el análisis de variantes genéticas humanas	26
3.3. Integración del flujo de trabajo en una aplicación Shiny interactiva: GenAgans	33
3.3.1. Funciones aisladas e instantáneas	33
3.3.2. Estructura y funciones recurrentes del servidor	34
3.3.3. Estructura y funciones recurrentes de la UI	36
3.3.1. Cajas de textos fijos	39
3.3.2. Cajas de carga de archivos.....	40
3.3.3. Cajas de texto reactivo.....	41
3.3.4. Botones de avance entre pestañas.....	43
3.3.5. Paneles de opciones para la personalización de los comandos del flujo de trabajo	45
3.3.6. Botones de inicio de procesos	48
3.3.7. Cajas de renderizado de resultados.....	51
3.3.8. Botones de descarga de archivos	52
3.3.9. Generación de un script personalizado.....	53

4. Resultados y discusión.....	54
5. Conclusiones.....	62
6. Limitaciones y perspectivas futuras.....	63
7. Referencias bibliográficas	64
8. Anexos	70
Anexo 1. Script para el análisis de variantes genéticas en humano	70
Anexo 2. Sistema de carpetas creado con el script	73
Anexo 3. Código para la definición de la UI del apartado “Inicio”	74
Anexo 4. Comparativa de uso entre ui.layout_columns() y ui.layout_column_wrap()	76
Anexo 5. Ejemplo de uso de ui.input_checkbox(), ui.input_numeric() ui.input_text() en el código UI para el apartado “2. Limpieza de las lecturas”.....	77
Anexo 6. Apariencia del panel de opciones del apartado “2. Limpieza de las lecturas”	78
Anexo 7. Código UI para las opciones del apartado “9. Efecto de las variantes”	79
Anexo 8. Apariencia de las opciones y el comando del apartado “9. Efecto de las variantes”	81
Anexo 9. Apariencia de las opciones y el comando del apartado “9. Efecto de las variantes” tras seleccionar todas las opciones	82
Anexo 10. Comparación entre la caja de resultados del apartado “2. Limpieza de las lecturas” contraída vs expandida.....	83
Anexo 11. Variación en el código servidor para la descarga de todos los archivos en el apartado “Resumen”	84
Anexo 12. Desplegables del apartado “Resumen”	85
Anexo 13. Código del servidor para el script personalizado	86
Anexo 14. UI y prueba de funcionalidad de GenAgans	90
Anexo 15. Medidas de control en la UI	108

Listado de abreviaturas

ARNm	ARN mensajero
BAM	Archivo binario de alineamiento
GUI	Interfaz gráfica de usuario
Indels	Inserciones y delecciones de fragmentos de ADN
Inputs	Datos de entrada
kpb	Kilo pares de bases
NGS	Tecnologías de secuenciación de próxima generación
ONT	Oxford Nanopore Technologies
Outputs	Datos de salida
PacBio	Pacific Biosciences
RG	Grupos de lectura
SAM	Archivo de texto de alineamiento de secuencias
SNVs	Variantes de nucleótido único
SVs	Variantes estructurales
TS	Secuenciación dirigida
UI	Interfaz de usuario
VCF	Archivo con formato de llamada de variantes
WES	Secuenciación de los exomas
WGS	Secuenciación del genoma completo

Índice de tablas

Tabla 1. Herramientas instaladas en el ambiente de Conda.....	25
Tabla 2. Opciones adicionales empleadas en Fastp y su utilidad.....	28
Tabla 3. Opciones adicionales empleadas en FreeBayes y su utilidad.....	30
Tabla 4. Opciones adicionales empleadas en VCFtools y su utilidad.....	31
Tabla 5. Opciones adicionales empleadas en VEP y su utilidad.....	32

Índice de gráficos y figuras

Figura 1. Gráfica de evolución de la cantidad de información producida por carrera de secuenciación (en kpb) respecto a los años. Modificada de (1).....	12
Figura 2. Esquema del flujo de trabajo típico de un estudio de análisis de variantes. Extraída de (2).....	17
Figura 3. Flujo de trabajo estándar para la identificación y predicción del efecto de las variantes genéticas presentes en muestras secuenciadas. Modificada de (3).....	22
Figura 4. Esquema del funcionamiento del código de una aplicación Shiny.....	33
Figura 5. Código para la creación de la carpeta temporal, definición de una variable con su ruta y cambio del directorio actual a la carpeta temporal.....	34
Figura 6. Código de definición del servidor y de las variables globales reactivas para el apartado 1 y 2.....	34
Figura 7. Código para la definición reactiva de los comandos del apartado 1.....	35
Figura 8. Interfaz de la pestaña “Inicio” de la aplicación previa carga de archivos.....	36
Figura 9. Código para la definición de la estructura general de la UI y las distintas pestañas que la componen.....	37
Figura 10. Recorte del código para la definición de la UI de la pestaña “Inicio”.....	38
Figura 11. Código del servidor para la definición del texto fijo a mostrar en la introducción..	39
Figura 12. Código para la definición de las cajas de carga de archivos del apartado “Inicio”.....	40
Figura 13. Ejemplo del contenido de la variable “input.archivo1()”.....	41
Figura 14. Código en el servidor para la función de carga de archivos del archivo 1.....	41
Figura 15. Interfaz de la pestaña 1 de GenAgans tras cargar los archivos y ejecutar el comando.....	42
Figura 16. Código en el servidor para la caja de texto del comando del apartado 1.....	42
Figura 17. Código en la UI para la definición del botón “EMPEZAR” del apartado “Inicio” ...	43

Figura 18. Código en el servidor para la definición del botón “EMPEZAR” del apartado “Inicio”.....	44
Figura 19. Código en el servidor para la actualización del botón “siguiente” del apartado 1.....	44
Figura 20. Código en el servidor para producir el cambio de la pestaña de “Inicio” a la pestaña del apartado 1.....	45
Figura 21. Código del servidor para la actualización de las variables globales de definición reactiva según las opciones que marque el usuario.....	47
Figura 22. Código para la definición reactiva del comando VEP en función de las opciones que marque el usuario.....	48
Figura 23. Código UI para el botón de inicio de ejecución del comando del apartado 9.....	48
Figura 24. Código en el servidor para la ejecución de los comandos del apartado 9.....	50
Figura 25. Código en el servidor para el ajuste de formato de la tabla de resultados de VEP.....	51
Figura 26. Código UI para la caja de resultados de la tabla de VEP en el apartado 9.....	51
Figura 27. Código en el servidor para mostrar en la UI el resultado de la tabla de VEP en el apartado 9.....	52
Figura 28. Código UI para el botón de descarga de resultados del apartado 9.....	52
Figura 29. Código en el servidor para la descarga de los archivos generados en el apartado 9.....	52
Figura 30. Código UI para los desplegables del apartado “Resumen”.....	53
Figura 31. Esquema representativo de la interfaz y el flujo de trabajo de GenAgans desde el apartado “Inicio” hasta el apartado “5. Análisis del mapeo”.....	55
Figura 32. Esquema representativo de la interfaz y el flujo de trabajo de GenAgans para el apartado “6. Limpieza de duplicados” y el apartado “7. Llamada de variantes”.....	58
Figura 33. Esquema representativo de la interfaz y el flujo de trabajo de GenAgans desde el apartado “8. Filtrado de variantes” hasta el apartado “Resumen”.....	60

Figura A2. Sistema de carpetas creado automáticamente tras ejecutar el script.....	73
Figura A4. Comparativa entre la disposición de los componentes del panel de opciones de RG en función del uso de ui.layout_columns() (a) o ui.layout_column_wrap() (b).....	76
Figura A5. Código UI para definir el panel de opciones del apartado 2.....	77
Figura A6. UI para el panel de opciones del apartado 2.....	78
Figura A7.1. Código UI para el panel de opciones del apartado 9. #1.....	79
Figura A7.2. Código UI para el panel de opciones del apartado 9. #2.....	80
Figura A8. UI para el panel de opciones del apartado 9 sin seleccionar opciones.....	81
Figura A9. UI para el panel de opciones y el comando del apartado 9 seleccionando todas las opciones.....	82
Figura A10. UI para la caja de resultados del apartado 2.....	83
Figura A11. Código en el servidor para la descarga de todos los archivos generados en el apartado “Resumen”.....	84
Figura A12. UI para los desplegables del apartado “Resumen”	85
Figura A14.1. UI para el apartado “Inicio” con los archivos cargados.....	90
Figura A14.2. UI para el apartado 1 con los resultados contraídos.....	90
Figura A14.3. UI para el apartado 1 con uno de los resultados expandidos.....	91
Figura A14.4. UI para el apartado 2 con las opciones seleccionadas.....	91
Figura A14.5. UI para el apartado 2 con las opciones seleccionadas y el resultado contraído.....	92
Figura A14.6. UI para el apartado 2 con el resultado expandido.....	92
Figura A14.7. UI para el apartado 3 con los archivos cargados.....	93
Figura A14.8. UI para el apartado 4 con los archivos cargados.....	93
Figura A14.9. UI para el apartado 5 con los resultados comprimidos.....	94
Figura A14.10. UI para el apartado 5 con los resultados expandidos.....	95

Figura A14.11. UI para el apartado 6 con los resultados del marcaje de duplicados contraídos.....	95
Figura A14.12. UI para el apartado 6 con los resultados del marcaje de duplicados expandidos.....	96
Figura A14.13. Mensaje de advertencia sobre los RG en el apartado 6.....	96
Figura A14.14. UI para el apartado 6 con el comando de comprobación de RG ejecutado y el de adición o reemplazo de RG seleccionado.....	97
Figura A14.15. UI para el apartado 7 con los resultados contraídos.....	98
Figura A14.16. UI para el apartado 7 con los resultados expandidos.....	99
Figura A14.17. UI para el apartado 8 con los resultados contraídos.....	100
Figura A14.18. UI para el apartado 8 con los resultados expandidos.....	101
Figura A14.19. UI para el apartado 9 con los resultados contraídos.....	102
Figura A14.20. UI para el apartado 9 con los resultados de la tabla expandidos. #1.....	103
Figura A14.21. UI para el apartado 9 con los resultados de la tabla expandidos. #2.....	103
Figura A14.22. UI para el apartado 9 con los resultados de la tabla expandidos. #3.....	104
Figura A14.23. UI para el apartado 9 con los resultados de la tabla expandidos. #4.....	104
Figura A14.24. UI para el apartado 9 con los resultados de la tabla expandidos. #5.....	105
Figura A14.25. UI para el apartado 9 con los resultados generales expandidos.....	105
Figura A14.26. UI del apartado “Resumen” con el resumen de comandos desplegado....	106
Figura A14.27. UI del apartado “Resumen” con el script personalizado desplegado.....	106
Figura A14.28. UI del apartado “Resumen” con el script personalizado expandido.....	107
Figura A14.29. UI del apartado “Resumen” con el resumen de resultados desplegado....	107
Figura A15.1. UI del apartado “Inicio” sin archivos cargados.....	108
Figura A15.2. UI del apartado 1 sin archivos cargados.....	108
Figura A15.3. UI del apartado 1 con archivos cargados previa ejecución del comando....	109

Resumen y palabras clave

En las últimas décadas, las técnicas de secuenciación han experimentado una gran revolución, generando diariamente enormes cantidades de datos. Las mejoras en el coste/eficiencia de dichas técnicas han impulsado la aplicación de la genómica a la clínica, destacando especialmente la identificación y predicción del efecto de las variantes genéticas humanas para su uso en medicina personalizada. Esto ha llevado a numerosos clínicos y científicos a enfrentar el desafío de analizar grandes volúmenes de datos biológicos sin tener conocimientos previos en bioinformática. En el presente trabajo se ha desarrollado GenAgans, una aplicación que facilita el desarrollo del flujo de trabajo para el análisis de las variantes genéticas humanas a usuarios con poca o ninguna experiencia en bioinformática. Mediante el uso de Shiny para Python y de herramientas bioinformáticas consolidadas, se ha logrado que GenAgans guíe al usuario a lo largo de un flujo de trabajo robusto de forma interactiva, intuitiva, personalizable y reproducible.

Palabras clave: aplicación, indel, interactiva, intuitiva, medicina personalizada, Python, secuenciación, Shiny, SNV, VEP.

1. Introducción

1.1. La bioinformática y la era del *Big Data*

Desde que en 1962 Margaret Dayhoff y Robert Ledley desarrollaron el primer programa bioinformático para la determinación de la estructura primaria de las proteínas a partir de pequeños fragmentos peptídicos (4), la tecnología de secuenciación ha avanzado de forma exponencial (5).

En 1977 Sanger publicó el primer método para la determinación de secuencias de ADN (6). El método de secuenciación de Sanger se basa en el uso de nucleótidos que carecen del grupo 3'-OH (dideoxinucleótidos), lo provoca que no se pueda crear un enlace fosfodiéster con el siguiente nucleótido durante la síntesis de ADN. Así, con estos nucleótidos marcados radioactivamente se observa el último nucleótido introducido en la secuencia de ADN. Al repetir este proceso varias veces, finalmente se obtiene la secuencia de nucleótidos del fragmento de ADN sintetizado. Sanger no solo marcó el inicio de la secuenciación de primera generación, si no que se sigue utilizando hoy en día (5,7). Aunque la secuenciación de Sanger es ampliamente reconocida, el método de secuenciación de Maxam-Gilbert también es un método de secuenciación de primera generación (5,8). La razón por la que el método de Sanger prevaleció y el de Maxam-Gilbert dejó de emplearse radica en la toxicidad de los químicos necesarios para la secuenciación de Maxam-Gilbert y la simplicidad del método de Sanger en comparación con el anterior. Además, posteriormente, la toxicidad del método de Sanger se redujo todavía más con la sustitución del marcaje radioactivo por marcaje fluorescente (9).

El método de Sanger fue el principal método de secuenciación durante décadas y permitió la secuenciación del genoma del bacteriófago $\Phi X174$, el primer genoma de ADN secuenciado (5,10). Sin embargo, su baja escalabilidad y alto coste por genoma secuenciado llevó a la necesidad de encontrar nuevos métodos que permitiesen aumentar la eficiencia y reducir el coste de secuenciación. De esta forma, surgieron las tecnologías de secuenciación de segunda generación, también conocidas como tecnologías de secuenciación de próxima generación (*Next Generation Sequencing* o NGS) (9).

Estas nuevas tecnologías se empezaron a emplear a principios de los 2000 y permitieron por primera vez la secuenciación en paralelo. Esto supuso una revolución en el campo de la secuenciación, ya que con estas tecnologías se puede realizar el análisis de miles a millones de moléculas de ADN en una misma carrera, lo que conllevó una mejora en la rapidez y eficiencia del proceso de secuenciación y una gran reducción en el coste por genoma secuenciado (5,9,11). Algunas de las tecnologías de segunda generación que se desarrollaron son Roche 454, basada en la detección de la liberación de pirofosfato (pirosecuenciación); Ion Torrent de ThermoFisher, basada en la detección de iones de hidrógeno; Illumina MiSeq y Hiseq, basada en la detección de la fluorescencia emitida por nucleótidos modificados (terminadores reversibles) al incorporarse a la cadena de ADN en síntesis; y SOLiD, basada en la ligación de oligonucleótidos (9,11).

Si bien las tecnologías de secuenciación de segunda generación fueron una gran revolución por sus mejoras en coste y eficiencia respecto a las de primera generación, todavía tenían ciertas limitaciones que afrontar. Tanto las tecnologías de primera generación como las de segunda solo permiten la secuenciación de fragmentos cortos de ADN. Además, las de segunda generación requieren de pasos de preparación de las muestras y amplificación por PCR que limitan su coste/eficiencia y aumentan el riesgo de introducir errores. Por este motivo, se siguieron desarrollando nuevos métodos hasta que sobre el año 2010 surgieron las tecnologías de secuenciación de tercera generación (9).

Estas nuevas tecnologías se caracterizan por permitir la secuenciación de fragmentos largos de ADN y no requerir el paso de amplificación por PCR (12). Las dos plataformas pioneras en esta nueva generación de secuenciadores fueron Pacific Biosciences (PacBio) y Oxford Nanopore Technologies (ONT). La tecnología de PacBio se basa en la secuenciación de una sola molécula en tiempo real (SMRT, del inglés *Single-Molecule Real-Time sequencing*) mediante la detección de luz cuando un nuevo nucleótido es añadido a la secuencia de ADN en síntesis. Mientras que la de Oxford Nanopore se basa en la detección de la señal eléctrica que se produce cuando los nucleótidos de una secuencia de ADN (o ARN) pasan de forma lineal a través de un nanoporo (12,13).

Así, la llegada de las tecnologías de segunda y tercera generación redujo significativamente el coste de secuenciación, acompañada de un aumento en la eficacia gracias a la secuenciación masiva en paralelo y a las mejoras en la velocidad del proceso. Todo esto

condujo finalmente a un aumento exponencial de la cantidad de datos generados (14). Por ejemplo, con la tecnología de primera generación de Sanger a principios de los 2000 se podían obtener hasta 115 kpb (kilo pares de bases) en un día (10^2 kpb/día), mientras que en 2010 los secuenciadores masivos ya podían producir billones de kilo pares de bases de datos por día (10^{14} kpb/día) (Fig. 1) (1).

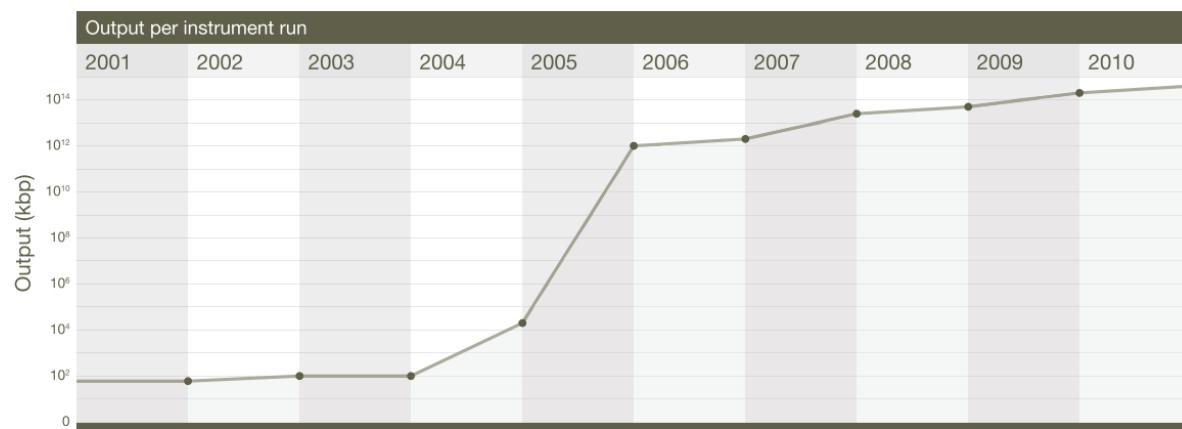


Figura 1. Gráfica de evolución de la cantidad de información producida por carrera de secuenciación (en kpb) respecto a los años. Se observa un aumento brusco entre el año 2004 y el 2006, que luego sigue aumentando de forma más paulatina. Modificada de (1).

Se entró entonces en la llamada era del *Big Data* caracterizada por la urgente necesidad de manejar y analizar grandes cantidades de información. Por lo que, en torno al año 2010, nació el rol del bioinformático como el especialista en el uso o desarrollo de herramientas para analizar los datos biológicos generados. Sin embargo, la definición de este término sigue siendo objeto de debate en la actualidad (5).

El desarrollo de dichas mejoras técnicas en el campo de la secuenciación y otros como la espectroscopía y espectrometría de masas, llevó a la especialización de algunas tecnologías en objetivos moleculares concretos, produciéndose así el nacimiento de las ciencias ómicas como la genómica, transcriptómica, proteómica y metabolómica (15). En la clínica, el uso individual o conjunto de estas ciencias permite identificar nuevos marcadores biomoleculares para el diagnóstico temprano de enfermedades, encontrar nuevas dianas para tratamientos de enfermedades complejas y estimar la probabilidad de que cierto individuo desarrolle una determinada enfermedad (16).

1.2. La genómica y las variantes genéticas en la biomedicina

La Organización Mundial de la Salud define la genómica como “el estudio del conjunto completo de genes (el genoma) de los organismos, su función y la interacción entre ellos y el ambiente” (17).

Gracias a los avances en las técnicas de secuenciación, la genómica aplicada a la clínica ha cobrado especial relevancia (18,19). En este sentido, facilita la predicción y prevención de enfermedades con base genética. También permite administrar medicamentos en función de la genética del paciente (farmacogenómica), de forma que estos resulten más efectivos y se reduzca la probabilidad de efectos adversos. Además, agiliza la recopilación de nueva información genética mediante la secuenciación masiva, lo que se traduce en el descubrimiento de nuevas dianas terapéuticas (2). Asimismo, en oncología, la genómica ha permitido la caracterización genética (genotipado) de distintos tipos de tumores. Esto ha posibilitado comprender las bases genéticas de los diferentes tipos de cáncer, encontrar biomarcadores que indiquen la respuesta del tumor al fármaco administrado y desarrollar terapias dirigidas específicamente contra el tumor (20).

Cabe destacar que la gran mayoría de las aplicaciones de la genómica en la clínica se basan en la detección y análisis de variantes genéticas (2,20). Por lo tanto, para comprender mejor la importancia de estos estudios clínicos, a continuación se explica qué son las variantes genéticas y los principios en los que se basan este tipo de investigaciones.

1.2.1. Tipos de variantes genéticas

Las variantes genéticas son cambios en una secuencia de ADN que hacen que esta difiera de la secuencia ADN de un genoma de referencia o consenso (18).

Desde un punto de vista estructural existen 3 tipos de variantes genéticas. Las dos más comunes son las variantes de nucleótido único (SNVs) y las inserciones/delecciones de fragmentos de ADN (indels). Las SNVs son variantes que implican el cambio en un único nucleótido de la secuencia de ADN secuenciada respecto a la de referencia. Por su parte, los indels son variantes que implican la inserción o delección de unos pocos nucleótidos en

la secuencia de ADN secuenciada respecto a la de referencia. El tercer tipo son las variantes estructurales (SVs). Estas son variantes poco frecuentes que implican inserciones, delecciones, duplicaciones, inversiones y translocaciones de grandes segmentos de ADN (21).

Desde el punto de vista del efecto que tienen las variantes genéticas sobre la codificación y, por tanto, sobre la función de las proteínas, se pueden clasificar principalmente en variantes sinónimas y no sinónimas (21).

Las variantes sinónimas son aquellas cuyo cambio en la secuencia de ADN produce tripletes de nucleótidos que codifican para el mismo aminoácido que la secuencia de referencia. Por tanto, las variantes sinónimas no suelen afectar a la función de la proteína, ya que no producen un cambio en la secuencia aminoacídica (21). Sin embargo, en algunos casos pueden afectar a la eficiencia de la transcripción y la traducción, producir inestabilidad en el ARN mensajero (ARNm) o alterar el splicing del ARNm durante la transcripción. Así pues, las variantes sinónimas pueden resultar patogénicas en algunos casos (21,22). Dentro de este tipo de variantes se encuentran las SNVs que no producen un cambio de aminoácido y los indels que no alteran la pauta de lectura y que tampoco producen un cambio de aminoácido (21).

Por otro lado, las variantes no sinónimas son aquellas cuyo cambio en la secuencia de ADN sí que produce cambios en la secuencia aminoacídica. De esta forma, se altera la síntesis de ARNm y, por tanto, de la proteína resultante. Debido a dichas alteraciones, estas variantes suelen producir patologías (21,23).

Estos cambios pueden ser producidos por diferentes razones, por lo que se distinguen diferentes tipos de variantes dentro de la categoría de variantes no sinónimas (21):

En primer lugar, las variantes sin sentido (también conocidas como *nonsense*) son aquellas SNVs cuyo cambio de nucleótido produce un triplete que codifica un codón de parada prematuro. Esto provoca que la transcripción finalice antes de lo que esperado y que se produzca un ARNm incompleto que se traduce en una proteína truncada. Las proteínas truncadas generalmente se degradan por lo que estas variantes propician la pérdida de función de la proteína (21).

En segundo lugar, las variantes de cambio de sentido (también conocidas como *missense*) son aquellas SNVs cuyo cambio de nucleótido produce un triplete que codifica para un aminoácido diferente al de la secuencia de referencia, lo que provoca la adición de un aminoácido diferente durante la síntesis de la proteína. Esto puede tener graves consecuencias en la funcionalidad de la proteína. Así pues, se pueden encontrar variantes que provoquen que la proteína adquiera una función diferente a la normal (variantes de ganancia de función), y variantes que provoquen que la proteína pierda su función (variantes de pérdida de función) (21,24).

En tercer lugar, las variantes de cambio de pauta de lectura (también conocidas como *frameshift*) son aquellas indels que producen un cambio en la pauta de lectura del ADN durante la transcripción. De esta forma, los tripletes de nucleótidos del ADN que se identifican son diferentes a los que se identificarían en la secuencia original. Esto ocasiona que se sintetice un ARNm que codifica una serie de aminoácidos distinta y que, generalmente, se traduzca una proteína truncada. Por tanto, este tipo de variantes también suelen conllevar la pérdida de función (21).

Por último, las indels sin cambio pauta de lectura (también conocidos como *in-frame indels*) son indels que no modifican la pauta de lectura de los tripletes de nucleótidos del ADN, pero sí que afectan a la secuencia de aminoácidos que codifica. Aquí hay dos escenarios posibles. Si se trata de una delección, se pierden tripletes de nucleótidos, por lo que faltarán aminoácidos en la proteína sintetizada y probablemente se trunque. Si se trata de una inserción, se puede dar el efecto de las variantes sin sentido si se codifica un codón de parada prematuro, o el efecto de las variantes de cambio de sentido si simplemente se produce un cambio en la secuencia de aminoácidos que codifican los tripletes. Por tanto, estas variantes pueden provocar tanto pérdida como ganancia de función dependiendo de cómo alteren la secuencia de nucleótidos, siendo causantes de diferentes patologías (21).

1.2.2. Análisis de variantes

Para la detección de variantes genéticas hay tres aproximaciones principales: **1)** La secuenciación del genoma completo (WGS, del inglés *Whole Genome Sequencing*). Para ello se fragmenta el DNA de la muestra y se secuencian todos los fragmentos obtenidos. **2)** La secuenciación de los exomas (WES, del inglés *Whole Exome Sequencing*), es decir, de las zonas del genoma que contienen genes que codifican proteínas. Para ello se amplifican únicamente los exones y se secuencian. **3)** La secuenciación dirigida (TS, del inglés *Targeted Sequencing*) de regiones concretas del genoma. Para ello, en primer lugar, se identifican regiones de interés. Luego existen dos aproximaciones: el enriquecimiento mediante amplicones y la captura híbrida. En la primera, se emplean cebadores específicos para amplificar las regiones de interés. En la segunda, el ADN se fragmenta y se hace pasar por un soporte con oligonucleótidos complementarios a la secuencia de interés, permitiendo que los fragmentos correspondientes sean capturados. Luego, estos fragmentos son enriquecidos y amplificados mediante PCR. Finalmente, en ambas aproximaciones se secuencian las regiones amplificadas ([Fig. 2](#)) (2,25).

Tras la secuenciación de los fragmentos de ADN por cualquiera de las tres aproximaciones mencionadas, el siguiente paso consiste en su análisis con el objetivo de identificar las variantes presentes y su contexto genómico. Las secuencias de nucleótidos obtenidas tras la secuenciación se denominan “lecturas”. Las lecturas se alinean en función de su posición respecto a un genoma de referencia en un proceso llamado “mapeo”. Posteriormente, las lecturas mapeadas se contrastan con un genoma de referencia para identificar las variantes genéticas en un proceso llamado “llamada de variantes”. Tras identificar las variantes, se determina su contexto genómico, es decir, se recopilan metadatos sobre ellas, como su localización en el genoma, tipo de variante o frecuencia alélica, en un proceso llamado “anotación de variantes” (2,3,26).

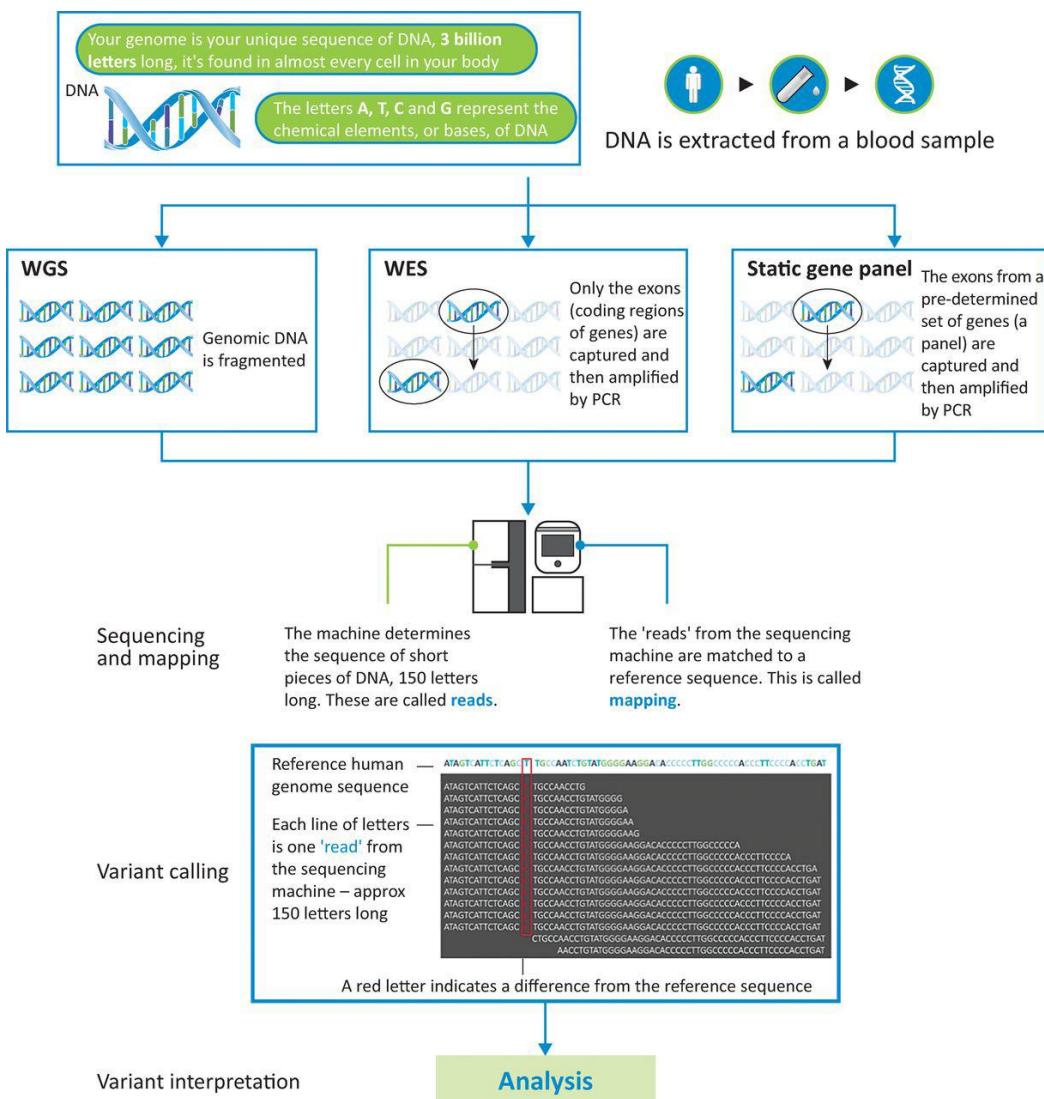


Figura 2. Esquema del flujo de trabajo típico de un estudio de análisis de variantes. En primer lugar, se extrae el ADN de la muestra. Luego se secuencia siguiendo la aproximación más adecuada para el estudio: WGS, WES o TS. Seguidamente se realiza el mapeo de las lecturas contra el genoma de referencia. Por último, se hace la llamada de variantes para obtener las variantes presentes en la muestra y se analizan. Extraída de (2).

Una vez se ha realizado la anotación de las variantes encontradas, el siguiente paso consiste en determinar su relevancia clínica y funcional. Por un lado, puede que las variantes encontradas hayan sido observadas y estudiadas previamente. En estos casos, se puede realizar una búsqueda en bases de datos especializadas para obtener más información sobre las variantes encontradas (26–29). Algunas de las bases de datos más empleadas para este propósito son ClinVar (30), OMIM (31), dbSNP (32), y Ensembl (33). En función de los estudios disponibles en las bases de datos, estas variantes conocidas se clasifican en: “patogénicas”, “probablemente patogénicas”, “probablemente benignas” o “benignas”. Donde “probablemente” indica que se ha observado con un 90% de certeza que

la variante es patogénica o benigna. Sin embargo, en la mayoría de los casos las variantes encontradas no han sido observadas previamente o no hay suficientes estudios que respalden su efecto y, por tanto, su impacto es desconocido. Estas últimas son clasificadas como “variantes de significado incierto” (VUS, del inglés Variants of Uncertain Significance) (26–29).

Dado que las VUS no pueden ser respaldadas por las evidencias científicas presentes en las bases de datos, es importante poder predecir su impacto mediante otros métodos (29,34). Desde el punto de vista bioinformático, se han desarrollado algoritmos para la predicción de los efectos de las variantes. Estos se basan generalmente en la localización de la variante, su conservación evolutiva y las consecuencias de la sustitución del aminoácido en la estructura y función de la proteína (34).

Por ejemplo, algoritmos como SpliceAI predicen el impacto que pueden tener las variantes, tanto sinónimas como no sinónimas, sobre el splicing del ARNm en función de su localización (35). Por otra parte, SIFT y PolyPhen son dos algoritmos ampliamente utilizados para predecir el impacto de variantes no sinónimas sobre la estructura y función de una proteína debido a un cambio de aminoácido (36,37). En el caso de SIFT, este se basa en la conservación evolutiva. De esta forma, se fundamenta en la premisa de que las regiones menos conservadas de la proteína tienden a ser más tolerantes a las variaciones, por lo que un cambio de aminoácido tendrá una menor probabilidad de afectar a su función, y viceversa (36). En el caso de PolyPhen, se basa en la predicción de la estructura y función que tendrá la proteína tras el cambio de aminoácido en función de las características bioquímicas que se vean alteradas (37).

Estos algoritmos no deben sustituir las evidencias experimentales, ya que son solo predictores *in silico* y sus resultados deben ser interpretados con cautela (38). Aun así, son especialmente útiles para tener una primera aproximación de los posibles efectos de las VUS. De esta forma, resultan de gran utilidad para discernir las variantes con mayor probabilidad de ser causantes de una enfermedad, guiando futuros estudios para su reclasificación (39). Por ejemplo, el uso combinado de estos algoritmos se ha empleado para identificar las VUS con mayor probabilidad de causar distrofia retiniana hereditaria, permitiendo su estudio en profundidad y futura reclasificación (40).

Aun así, la aplicación de estos algoritmos no se limita únicamente a las VUS, también son útiles para realizar una evaluación general de las variantes ya clasificadas. De esta forma, las predicciones obtenidas mediante los algoritmos pueden respaldar dichas clasificaciones, otorgándoles un nivel adicional de evidencia. Predicciones que validen o refuerzen la clasificación de una variante pueden ser especialmente importantes en casos de variantes clasificadas como “probablemente patogénicas” o “probablemente benignas”, es decir, cuando todavía no hay suficiente evidencia científica como para clasificar una variante como “patogénica” o “benigna” (39).

1.2.3. Relevancia de la predicción de variantes genéticas en la práctica clínica

Dadas las implicaciones que tiene la identificación y predicción del efecto de las variantes, su uso se ha vuelto cada vez más habitual en el ámbito clínico (41).

Por una parte, su uso combinado con estudios de asociación del genoma completo (GWAS, del inglés *Genome-Wide Association Studies*) ha promovido el desarrollo de la medicina personalizada y terapias génicas (42). Los GWAS son un tipo de estudio en el que se examina el genoma completo para identificar las regiones del ADN cuyas variaciones se asocian a una mayor predisposición a desarrollar una enfermedad (43). Así, los estudios GWAS identifican aquellas variantes que son más prevalentes en individuos con cierta enfermedad al compararlos con individuos sanos. Luego, la predicción del efecto de las variantes ayuda a identificar qué variantes de las identificadas con GWAS tienen mayor probabilidad de ser las causantes de la enfermedad por su carácter patogénico. De esta forma, se reduce el número de variantes candidatas y se prioriza su investigación. Cuando finalmente se ha validado experimentalmente que una de estas variantes es causante de la enfermedad, se puede emplear para la prevención, el diagnóstico y pronóstico en aquellos pacientes que presenten esta variante y/o como diana para el desarrollo de nuevos tratamientos como terapias dirigidas y terapias génicas (42).

De forma similar, la identificación y predicción del efecto de las variantes resulta de gran importancia en la medicina predictiva y preventiva. Su utilidad en estos casos radica en la detección temprana de variantes de riesgo de forma que se pueda asesorar sobre la susceptibilidad de padecer la enfermedad (medicina predictiva) y tomar medidas para evitar

o minimizar la probabilidad de desarrollarla (medicina preventiva). Por ejemplo, en el caso del cáncer de mama, la detección temprana de variantes en genes de riesgo como BRCA1 y BRCA2, permite el asesoramiento médico sobre posibles intervenciones o medidas para reducir el riesgo de llegar a desarrollar la enfermedad (44). Similarmente, la detección de variantes de riesgo y predicción del efecto de las VUS también se emplea en ámbito del consejo genético, especialmente en la evaluación del riesgo hereditario (41).

Por otra parte, la identificación de variantes ha cobrado gran relevancia en el campo de la farmacogenómica, ya que permite la prescripción de fármacos adaptados a las necesidades de cada paciente en función de las variantes que presente. En oncología de precisión se ha empleado la detección de variantes para identificar aquellos tumores que desarrollan resistencia ciertos tratamientos. Por ejemplo, en casos de cáncer de pulmón no microcítico (NSCLC), se ha observado que la aparición de la variante T790M del gen codificador para el receptor del factor de crecimiento epidérmico (EGFR) provoca la resistencia del tumor a tratamientos basados en inhibidores de la tirosina quinasa (TKIs) de primera y segunda generación, como el geftinib y el erlotinib. En estos casos, la identificación de esta variante permite la detección temprana de la resistencia a estos fármacos y, por tanto, el cambio a un tratamiento más efectivo (45).

1.3. Flujo de trabajo para la identificación y predicción del efecto de variantes genéticas

La identificación y predicción del efecto de las variantes es un campo muy relevante en genómica. Aunque existen muchos programas para este análisis, el flujo de trabajo sigue unos estándares preestablecidos y cuenta con una serie de pasos críticos como el alineamiento de las secuencias, la llamada de variantes y su anotación. A continuación se describen los pasos de un flujo de trabajo estándar para el análisis de las variantes, también llamado “flujo de trabajo para NGS” (Fig. 3) (3).

En primer lugar, se realiza un análisis de calidad de las lecturas contenidas en archivos FASTQ. Este es un archivo de texto que contiene las secuencias de nucleótidos obtenidas tras la secuenciación junto con sus indicadores de calidad. Seguidamente, se hace de un

filtrado de aquellas lecturas de baja calidad y/o se eliminan artefactos de secuenciación (3,46).

Posteriormente, las lecturas limpias se mapean contra el genoma de referencia. Este paso produce un archivo de alineamiento y mapeo de secuencias (SAM, del inglés *Sequence Alignment/Map*). El archivo SAM es un archivo de texto que contiene información sobre las lecturas mapeadas (3,46).

A continuación, el SAM se procesa para generar un archivo binario de alineamiento y mapeo (BAM, del inglés *Binary Alignment/Map*). El archivo BAM es la versión comprimida en binario del archivo SAM. Por tanto, ya no es humanamente legible, pero resulta menos pesado computacionalmente, por lo que su procesamiento es más eficiente. Una vez obtenido el BAM, se marcan o eliminan los duplicados para evitar que actúen como artefactos técnicos que alteren las frecuencias observadas (3,46).

Seguidamente, se realiza la llamada de variantes. Así se identifican las variantes presentes en las muestras comparando la secuencia del genoma de referencia con las secuencias de las lecturas alineadas previamente y presentes en el archivo BAM. En este proceso se genera un archivo con formato de llamada de variantes (VCF, del inglés *Variant Call Format*). Este archivo de texto almacena detalles sobre las variantes identificadas en la llamada de variantes. Las variantes del archivo VCF pueden ser filtradas en función de su calidad o profundidad de secuenciación, entre otros, para eliminar falsos positivos. Posteriormente, se anotan las variantes a partir del archivo VCF. Este proceso implica la recopilación de metadatos adicionales sobre las variantes presentes en el archivo VCF a partir de bases de datos, y la predicción de su efecto mediante algoritmos de predicción (3,46).

Finalmente, se obtiene un informe con todos los datos recopilados sobre las variantes identificadas. A partir de este informe, los expertos clínicos pueden extraer la información más relevante para el objetivo de su estudio y así tomar las decisiones oportunas (3,46).

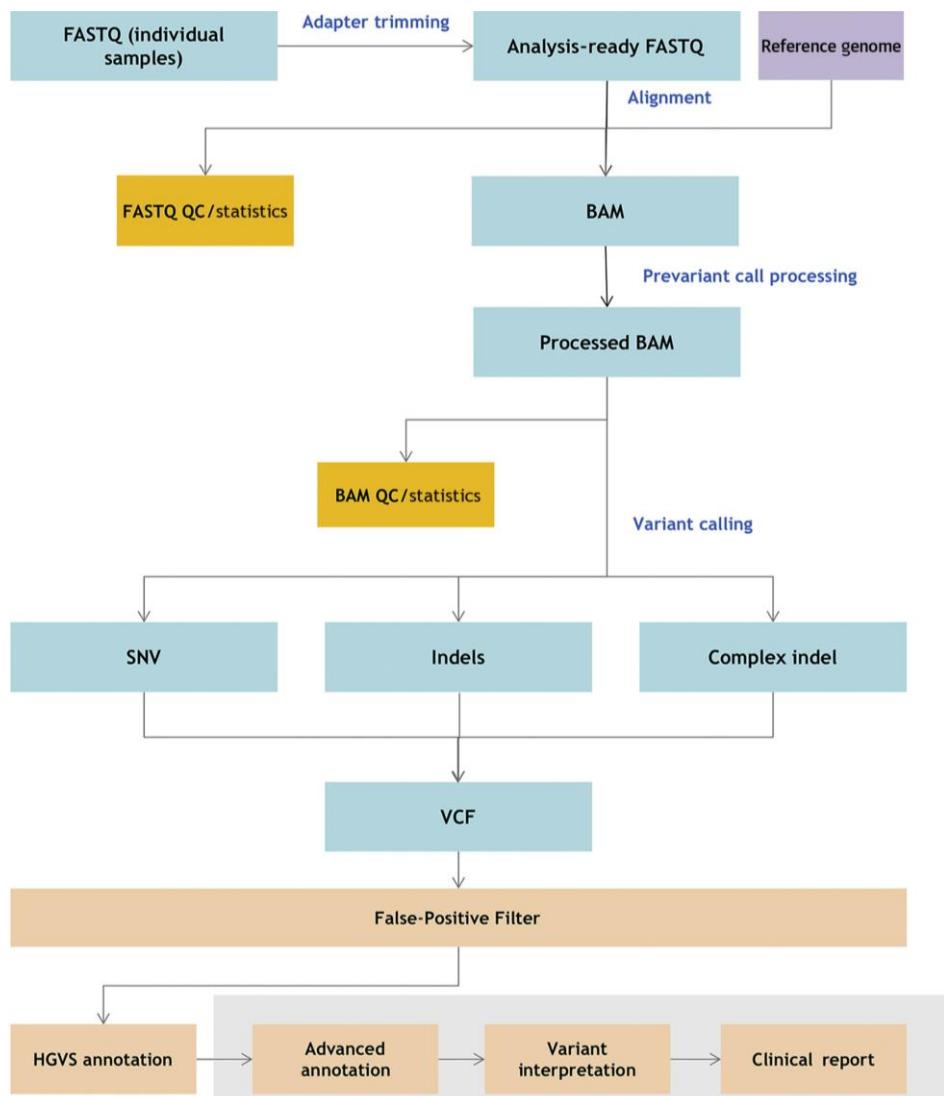


Figura 3. Flujo de trabajo estándar para la identificación y predicción del efecto de las variantes genéticas presentes en muestras secuenciadas. Los archivos FASTQ de lecturas se recortan para eliminar artefactos de secuenciación. Las lecturas del FASTQ limpio se mapean contra el genoma de referencia. Se procesa el archivo BAM generado. En la llamada de variantes se identifican las diferentes variantes presentes a partir del archivo BAM y se almacenan en un archivo VCF. El archivo VCF se puede filtrar para eliminar falsos positivos. Se realiza la anotación de las variantes. Se obtiene un informe clínico para su interpretación. Modificada de (3).

1.4. Desarrollo de flujos de trabajo bioinformáticos

Cada uno de los pasos de un flujo de trabajo como el previamente descrito requiere la implementación de algoritmos bioinformáticos. Estos algoritmos involucran conocimientos en programación y en diferentes disciplinas matemáticas como análisis funcional, optimización, topología y análisis numérico, entre otros. Esto hace que dichos algoritmos sean intrínsecamente complejos de entender e implementar (47).

Esta problemática ha impulsado el desarrollo de herramientas bioinformáticas que incorporan estos algoritmos y facilitan su aplicación. Aun así, incluso las herramientas bioinformáticas requieren de conocimientos básicos de programación para emplearlas. Además, cada herramienta puede haber sido desarrollada en un lenguaje de programación distinto y requerir unos datos de entrada (inputs) y datos de salida (outputs) diferentes con unos formatos concretos. Por esta razón, se han desarrollado diversas plataformas para facilitar su acceso a un público más general (48). A continuación se describen algunas de ellas:

Galaxy: presenta una interfaz gráfica de usuario (GUI, del inglés *Graphical User Interface*) interactiva y cuenta con más de 5500 herramientas bioinformáticas. No requiere de extensos conocimientos en programación, por lo que es ampliamente utilizada para el análisis de datos biomédicos en áreas como la genómica, la proteómica y la metabolómica (49).

Snakemake: no presenta GUI ya que está dirigida a usuarios con conocimientos de programación, concretamente en Python. Permite la creación, automatización y gestión de flujos de trabajo complejos, lo que le hace ideal para el análisis masivo de datos (50).

Taverna: presenta una GUI intuitiva que permite al usuario diseñar flujos de trabajo personalizados sin necesidad de escribir código (51).

Cada una de estas herramientas presenta un enfoque distinto para manejar la accesibilidad y complejidad de las herramientas bioinformáticas, adaptándose a diferentes niveles de experiencia y necesidades específicas (49–51).

1.5. Problemática presente y justificación del desarrollo de este trabajo

La bioinformática es un campo en auge con una alta demanda de expertos debido a las grandes cantidades de datos biológicos que se generan diariamente. Aun así, muchos clínicos y científicos carecen de competencias en bioinformática. Las encuestas indican que un 74% de los científicos de laboratorios de biología no tienen experiencia en programación. Esto genera una brecha entre los conocimientos biológicos y las habilidades computacionales necesarias para llevar a cabo un flujo de trabajo bioinformático (52).

Si bien los programas y herramientas para el desarrollo de flujos de trabajo son de gran utilidad para la comunidad científica, todos ellos requieren que el usuario tenga conocimientos previos en bioinformática. Ya sea para decidir qué herramientas usar, qué archivos de entrada requieren o qué pasos debe tener el flujo de trabajo. En el presente trabajo se aborda la necesidad de facilitar el acceso a las herramientas bioinformáticas a estudiantes, científicos y clínicos que requieran identificar y predecir el efecto de variantes genéticas en muestras humanas, pero que carezcan de una formación sólida en bioinformática. Potenciando así que sus conocimientos en biomedicina puedan trasladarse con mayor facilidad al campo de la bioinformática aplicada a la medicina personalizada.

2. Objetivos

El objetivo principal de este trabajo consiste en desarrollar una aplicación bioinformática que facilite el desarrollo de un flujo de trabajo para la identificación y predicción del efecto de variantes genéticas presentes en muestras humanas de una forma interactiva, intuitiva, personalizada y reproducible. De forma que el usuario se enfoque en aplicar los conocimientos propios de su especialidad sin tener conocimientos previos en programación. Para ello, se proponen los siguientes objetivos específicos:

1. Desarrollar una aplicación que facilite el flujo de trabajo para la identificación y predicción del efecto de las variantes genéticas en muestras de humanos.
 - a. Añadir diferentes opciones que permitan del desarrollo del flujo de trabajo de forma interactiva y personalizada.
 - b. Gestionar de forma automática los input y outputs de las diferentes herramientas implementadas para el flujo de trabajo.
 - c. Implementar la generación de un informe con las variantes genéticas identificadas en las muestras y sus respectivas anotaciones.
2. Proveer a la aplicación de una interfaz gráfica interactiva.
 - a. Desarrollar una interfaz intuitiva que habilite las diferentes etapas del flujo de trabajo de forma secuencial y controlada.
 - b. Desarrollar una interfaz capaz de generar y permitir la visualización de diferentes informes generados a lo largo del flujo de trabajo.
 - c. Agregar botones y funcionalidades para la descarga de todos los archivos generados.

- d. Desarrollar la generación y descarga de un script personalizado que contenga todos los comandos necesarios para replicar exactamente el flujo de trabajo realizado. Reuniendo así todas las opciones empleadas por el usuario en cada comando.
- 3. Generar un ambiente de Conda con todas las herramientas y dependencias necesarias para la ejecución de la aplicación. De forma que se facilite su instalación y correcto funcionamiento.

3. Metodología

3.1. Materiales

La aplicación se desarrolló mediante el editor de código VSCode conectado a un sistema Linux (distribución Ubuntu-22.04) mediante la extensión WSL2 (*Windows Subsystem for Linux 2*). En el sistema Linux se instaló Conda (versión 23.9.0) y se creó un ambiente con Python (versión 3.8.18). En él se instalaron las herramientas mostradas en la Tabla 1.

Tabla 1. Herramientas instaladas en el ambiente de Conda. Fuente: Elaboración propia.

Herramienta	Versión	Comando de instalación
bwa	0.7.17	conda install -c bioconda bwa=0.7.17
ensembl-vep	112.0	conda install ensembl-vep
faicons	0.2.2	pip install faicons
fastp	0.23.4	conda install -c bioconda fastp=0.23.4
fastqc	0.12.1	conda install -c bioconda fastqc=0.12.1
freebayes	0.9.21.7	conda install -c bioconda freebayes=0.9.21.7
gnuplot	5.4.8	conda install -c conda-forge gnuplot=5.4.8
pandas	2.0.3	conda install -c conda-forge pandas=2.0.3
picard	3.1.1	conda install -c bioconda picard=3.1.1
rtg-tools	3.12.1	conda install -c bioconda rtg-tools=3.12.1
samtools	1.19	conda install -c bioconda samtools=1.19
shiny	1.0.0	pip install shiny
vcftools	0.1.16	conda install -c bioconda vcftools=0.1.16

Dada la extensión del código de la aplicación desarrollada (+2800 líneas), este se encuentra disponible en el [repositorio GitHub](#) “GenAgans” con el nombre “app.py”. En él también se disponen todos los archivos requeridos para su funcionamiento como el archivo YML para la instalación del ambiente de Conda y un LEEME.txt con las instrucciones para su ejecución.

Se emplearon dos conjuntos de datos para la comprobación del funcionamiento de la aplicación desarrollada. El primer conjunto comprende dos FASTQ.gz con lecturas pareadas de una muestra de tejido pulmonar secuenciada mediante WES proveniente de una paciente con carcinoma de células escamosas de pulmón (SQCC), y un FASTA con el genoma de referencia humano completo GRCh37. El segundo conjunto de datos comprende dos FASTQ.gz con lecturas pareadas de una muestra de tejido pulmonar secuenciada mediante WES proveniente de un paciente con adenocarcinoma de pulmón (ADCA), y un FASTA con el genoma de referencia humano completo GRCh38. Las muestras corresponden a un estudio para la búsqueda de nuevos biomarcadores en NSCLC (53). Los archivos de lecturas se descargaron de la base de datos ENA en el proyecto “PRJNA734015” y corresponden a los *Run Accession* “SRR14695036” y “SRR14695039”, respectivamente (54). Los archivos de genoma de referencia se descargaron de la base de datos ENSEMBL y corresponden a los enlaces de descarga de “Homo_sapiens.GRCh37.dna.primary_assembly.fa.gz” (55) y “Homo_sapiens.GRCh38.dna.primary_assembly.fa.gz” (56), respectivamente.

3.2. Flujo de trabajo para el análisis de variantes genéticas humanas

El desarrollo de este trabajo se dividió en dos fases. En primer lugar, se escribió un script en Bash con todos los comandos necesarios para la automatización del flujo de trabajo para el análisis de variantes genéticas humanas ([Anexo 1](#)). El propósito de la creación de dicho script fue validar previamente el correcto funcionamiento de los comandos ejecutables en Bash necesarios para el flujo de trabajo, ya que estos constituyen la base funcional del mismo. Para su correcta ejecución solo requería lanzarse en una carpeta que contuviese los dos archivos de lecturas pareadas en formato FASTQ.gz y el archivo de genoma de referencia en formato FASTA. A partir de ese directorio padre, el script ejecutaba

automáticamente todo el flujo de trabajo y generaba una serie de subcarpetas donde se organizaban todos los archivos generados durante el proceso ([Anexo 2](#)).

En segundo lugar, tras verificar el correcto funcionamiento de los comandos necesarios para el flujo de trabajo mediante el script de Bash, se procedió al desarrollo de la aplicación interactiva: **GenAgans**. Para ello, los comandos validados se integraron en un conjunto de funciones de Python para darle forma a la interfaz y reactividad a las acciones del usuario.

A continuación se listan, por orden de uso, las herramientas y opciones empleadas en cada comando del flujo de trabajo. Las opciones se dividen en “básicas” y “adicionales”. Las básicas son aquellas que aparecen de forma predeterminada en GenAgans. Las adicionales son aquellas que aparecen de forma opcional para permitir la personalización del comando.

1. FastQC (57):

FastQC es una herramienta para la evaluación de la calidad de lecturas obtenidas mediante secuenciación masiva. Se empleó para realizar un análisis de la calidad de las lecturas contenidas en los archivos FASTQ.gz. Tras el análisis se genera un informe con los resultados de calidad para cada uno de los archivos. En este informe se incluyen datos estadísticos y gráficas sobre la distribución de la calidad de las lecturas, contenido en GC y N, longitud de las lecturas y presencia de adaptadores, entre otros. Dicho informe facilita la toma de decisiones sobre las opciones a aplicar en la siguiente etapa, la limpieza de lecturas. El directorio donde se guardan dichos informes se indicó con la opción básica -o.

2. Fastp (58):

Fastp es una herramienta para el filtrado y limpieza de lecturas obtenidas mediante secuenciación masiva. Se empleó para realizar el filtrado y la limpieza de las lecturas contenidas en los archivos analizados previamente con FastQC. Al finalizar su ejecución se obtiene un informe con datos estadísticos y gráficos que permiten evaluar la calidad de las lecturas tras su limpieza. Este informe resulta útil para decidir si es necesario o no repetir la limpieza con parámetros más estrictos antes de proceder a la siguiente etapa, el mapeo. Como opciones básicas se emplearon -i/-I y -o/-O para indicar los archivos de entrada y el directorio y nombre de los de salida, respectivamente. Además de -h para generar un informe en HTML y -j en JSON. Las opciones adicionales incluidas se listan en la [tabla 2](#).

Tabla 2. Opciones adicionales empleadas en Fastp y su utilidad. Fuente: Elaboración propia.

Opción adicional	Utilidad
--cut_tail	Recortar cierto número de bases de baja calidad al final de cada lectura.
--cut_front	Recortar cierto número de bases de baja calidad al inicio de cada lectura.
--cut_mean_quality	Recortar las bases de los extremos con una calidad menor a la indicada.
--detect_adapter_for_pe	Detectar y eliminar automáticamente adaptadores en lecturas pareadas.
--trim_poly_g	Recortar los artefactos de secuencias poli-G.
--trim_poly_x	Recortar las secuencias de homopolímeros como poli-A, poli-T, etc.
-l	Eliminar las lecturas más cortas que el número indicado como parámetro.

3. BWA (*Burrows-Wheeler Aligner*) (59,60):

BWA comprende un conjunto de herramientas para el mapeo de lecturas de ADN contra una secuencia de referencia.

a. index:

“bwa index” se empleó para indexar el genoma de referencia. Este comando genera archivos de índice necesarios para el mapeo a partir del genoma de referencia. Con la opción -p se especificó la carpeta y el prefijo para los archivos de índice.

b. mem:

“bwa mem” se empleó para mapear las lecturas filtradas y limpiadas previamente con Fastp contra el genoma de referencia indexado. Este comando genera un archivo de mapeo SAM. Se añadió la opción -a para que se reportaran todas las alineaciones posibles para cada lectura, en lugar de únicamente la mejor. También se usó la opción -o para indicar la carpeta y nombre del archivo de salida.

4. SAMtools (61):

SAMtools comprende un conjunto diverso de herramientas esenciales para el procesamiento y manejo de archivos de mapeo (SAM/BAM).

a. view:

“samtools view” se empleó para convertir el archivo SAM, generado con bwa mem, en BAM. En este comando se indicaron las siguientes opciones: -b para indicar que el formato de salida del archivo será BAM; -s para indicar que el archivo de entrada es SAM; y -o para indicar la carpeta y el nombre del archivo de salida.

b. sort:

“samtools sort” se empleó para ordenar las lecturas del archivo BAM en función de sus coordenadas genómicas. Este paso es esencial para el marcaje de duplicados posterior. Con la opción -o se indicó la carpeta y el nombre del archivo de salida.

c. stats:

“samtools stats” se empleó para generar un archivo .txt con estadísticas sobre el archivo BAM, es decir, sobre el mapeo realizado con bwa mem.

d. plot-bamstats:

“plot-bamstats” se empleó para generar un archivo HTML con gráficos a partir de las estadísticas contenidas en el archivo .txt generado con samtools stats. La opción -p se usó para especificar la carpeta y el prefijo del archivo HTML.

5. Picard (62):

Picard comprende un conjunto de herramientas avanzadas para el análisis y manipulación archivos de mapeo (BAM/SAM).

a. MarkDuplicates

“picard MarkDuplicates” se empleó para marcar las lecturas duplicadas presentes en el archivo BAM. De esta forma, se evita que estos artefactos de secuenciación afecten negativamente a las etapas posteriores como la llamada de variantes. Este paso requiere que el archivo BAM esté ordenado por coordenadas para funcionar eficientemente. Así las lecturas están agrupadas por sus localizaciones genómicas y se facilita la identificación de aquellas duplicadas. Tras su ejecución se genera otro BAM con los duplicados marcados. Con la opción --INPUT se especificó el archivo BAM de entrada; con --OUTPUT el archivo BAM de salida con los duplicados marcados; con --METRICS_FILE se generó un archivo de métricas sobre las lecturas duplicadas; y con --ASUME_SORTED True se indicó que el archivo BAM de entrada ya está ordenado por coordenadas.

b. AddOrReplaceReadGroups

“picard AddOrReplaceReadGroups” se empleó para añadir o reemplazar los grupos de lectura (RG del inglés *Read Groups*). Los RG son etiquetas identificativas de las lecturas presentes en el archivo BAM. Estas son necesarias para el correcto funcionamiento de la llamada de variantes. Las etiquetas necesarias son: el identificador del grupo de lecturas (especificado con la opción -RGID), identificador de la librería de secuenciación (-RGLB), plataforma de secuenciación (-RGPL), nombre de la muestra (-RGSM), y nombre de la

carrera (-RGPU). Este comando toma el BAM generado con picard MarkDuplicates y, tras añadir o reemplazar los RG, genera un nuevo BAM que contiene dichas etiquetas.

6. SAMtools:

a. faidx:

“samtools faidx” se empleó para generar un archivo de índice del genoma de referencia (.fai). Este archivo es necesario para la llamada de variantes.

b. index:

“samtools index” se empleó para generar un archivo de índice del archivo BAM (.bai) de las lecturas mapeadas con los duplicados marcados y los RG establecidos. Este archivo es necesario para la llamada de variantes.

7. FreeBayes (63):

FreeBayes es una herramienta para la detección de variantes genéticas a partir de archivos de mapeo (BAM). Se empleó para realizar la llamada de variantes. Tras su ejecución se generó un archivo VCF. Como opción básica se empleó -f para especificar el genoma de referencia indexado previamente, que es contra el que se llaman las variantes. También se especificó el archivo BAM indexado previamente y se redirigió la salida a un archivo VCF. Las opciones adicionales incluidas y su utilidad se listan en la [tabla 3](#).

Tabla 3. Opciones adicionales empleadas en FreeBayes y su utilidad. Fuente: Elaboración propia.

Opción adicional	Utilidad
--ploidy	Indicar la ploidía de las muestras.
--min-alternate-count	Indicar el número mínimo de lecturas que deben apoyar el alelo alternativo para que se considere como variante.
--min-alternate-fraction	Indicar la fracción mínima de lecturas que deben apoyar el alelo alternativo para que se considere como variante.
--min-mapping-quality	Indicar la calidad mínima de mapeo que debe tener una lectura para considerarla en la llamada de variantes.

8. RTG-Tools (64):

RTG-Tools es una herramienta para la comparación y evaluación de variantes genéticas en archivos VCF.

a. rtg vcfstats:

“rtg vcfstats” se empleó para generar un archivo .txt con estadísticas sobre las variantes genéticas presentes en el archivo VCF generado en llamada de variantes. Esta herramienta

se empleó tanto inmediatamente después de la llamada de variantes como tras el filtrado de variantes para evaluar el contenido de los VCF tras dichos procesos.

9. VCFtools (65):

VCFtools es una herramienta para la manipulación y análisis de archivos VCF. Se empleó para filtrar las variantes presentes en el archivo VCF generado en la llamada de variantes. Tras su ejecución se genera otro archivo VCF con las variantes que han pasado los filtros. Las opciones básicas empleadas fueron: --vcf para especificar el archivo VCF de entrada; -recode para indicar que se debe generar un nuevo archivo VCF con las variantes que pasaron los filtros; --recode-INFO-all para incluir en el nuevo VCF toda la información de los campos INFO del VCF original; y --out para especificar la carpeta y el nombre del nuevo VCF generado. Las opciones adicionales incluidas y su utilidad se listan en la [tabla 4](#).

Tabla 4. Opciones adicionales empleadas en VCFtools y su utilidad. Fuente: Elaboración propia.

Opción adicional	Utilidad
--keep-only-indels	Mantener únicamente los indels.
--maxDP	Descartar las variantes con una profundidad de cobertura mayor a la indicada.
--minDP	Descartar las variantes con una profundidad de cobertura menor a la indicada.
--minQ	Descartar aquellas variantes con una calidad inferior a la especificada.
--remove-indels	Eliminar todos los indels.

10. VEP (*Ensembl Variant Effect Predictor*) (66):

VEP es una herramienta para la anotación y predicción del efecto de las variantes genéticas a partir de archivos VCF. Se empleó para realizar la anotación de las variantes genéticas presentes en el archivo VCF filtrado. Tras su ejecución se genera un informe general con gráficas y estadísticas de las variantes anotadas y una tabla con las anotaciones específicas para cada una de las variantes. Las opciones básicas empleadas fueron: -i (--input_file) para especificar el archivo VCF de entrada; -o (--output_file) para especificar el archivo de salida; --database para usar la base de datos online de Ensembl para anotar las variantes; --species para definir la especie ("homo_sapiens"); --force_overwrite para sobreescribir el archivo de salida si ya existía; y --tab para que el archivo de salida sea en formato tabulado. Las opciones adicionales incluidas y su utilidad se listan en la [tabla 5](#).

Tabla 5. Opciones adicionales empleadas en VEP y su utilidad. Fuente: Elaboración propia.

Opción adicional	Utilidad
--af	Obtener frecuencia alélica global de variantes colocalizadas con la variante.
--assembly	Indicar el ensamblado del genoma humano a emplear (GRCh37 o GRCh38).
--check_existing	Obtener datos de variantes conocidas colocalizadas con la variante.
--clin_sig_allele	Obtener información conocida sobre la significancia clínica de las variantes.
--fields	Limitar y ordenar los campos a incluir en la tabla resultante.
--gene_phenotype	Obtener fenotipos asociados al gen en el que se encuentra la variante.
--hgvs	Obtener la nomenclatura HGVS (<i>Human Genome Variation Society</i>).
--polyphen	Predecir el impacto funcional de la variante mediante el algoritmo PolyPhen.
--protein	Obtener el identificador Ensembl de la proteína asociada a la variante.
--sift	Predecir el impacto funcional de la variante mediante el algoritmo SIFT.
--spdi	Obtener la nomenclatura SPDI (Secuencia/Posición/Delección/Inserción).
--symbol	Obtener símbolo del gen en el que se encuentra la variante.
--uniprot	Obtener referencias UniProt de proteínas colocalizadas con la variante.
--variant_class	Obtener la clase de variante según <i>Sequence Ontology</i> .

Adicionalmente, en GenAgans se añadió un espacio de entrada de texto en cada uno de los comandos con opciones adicionales para introducir manualmente cualquier opción extra que el usuario requiera. Esto permite adaptar todavía más los comandos de las etapas del flujo de trabajo que requieren personalización: limpieza de lecturas (Fastp), llamada de variantes (FreeBayes), filtrado de variantes (VCFtools) y predicción del efecto de las variantes (VEP).

La elección de estas herramientas se basó en su amplio uso y aceptación por parte de la comunidad científica, por lo que se consideran herramientas robustas que proporcionan resultados fiables y de calidad (67,68). Por tanto, en conjunto crean un flujo de trabajo consistente con una gran variedad de opciones para su adaptación y personalización.

3.3. Integración del flujo de trabajo en una aplicación Shiny interactiva: GenAgans

El código de GenAgans se escribió en Python, excepto por los comandos de Bash necesarios para el flujo de trabajo descritos previamente. El principal paquete empleado para su desarrollo fue Shiny para Python (69). Esta versión de Shiny contiene una gran variedad de funciones para facilitar la creación de aplicaciones interactivas en Python (70).

El código de una aplicación Shiny se divide esencialmente en dos partes: funciones de la interfaz usuario (UI) y funciones del servidor (server). Las funciones de la UI proporcionan una interfaz gráfica en la que el usuario puede realizar acciones. Cuando el usuario realiza una acción sobre la UI, el input de la acción hace reaccionar a las funciones del servidor. Los outputs de las funciones del servidor se reportan a las funciones de la UI, de forma que la UI se actualiza. Así, cuando el usuario realiza una acción sobre la UI, el servidor reacciona, procesa y responde, enviando la respuesta a la UI que se actualiza y muestra el cambio en la interfaz gráfica (Fig. 4).

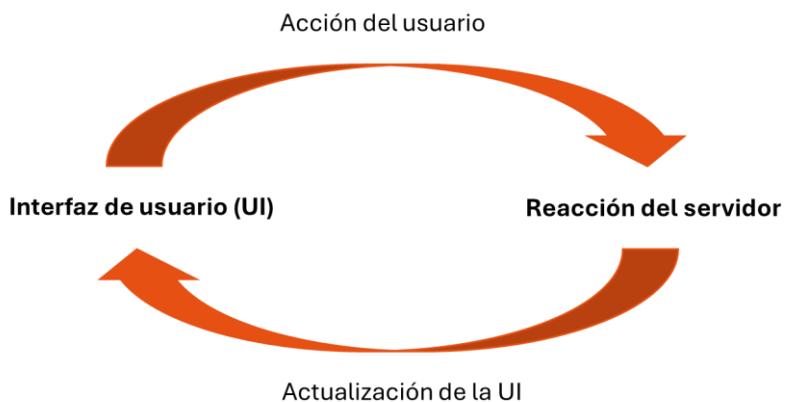


Figura 4. Esquema del funcionamiento del código de una aplicación Shiny. Fuente: Elaboración propia.

A continuación, se describe cómo integró el flujo de trabajo para el análisis de variantes genéticas en humanos en un código de dichas características.

3.3.1. Funciones aisladas e instantáneas

Como se trata de un flujo de trabajo diseñado para realizarse a partir de un directorio padre, se incluyeron 3 líneas de código esenciales que se ejecutan al lanzar la aplicación y que permiten su correcto funcionamiento. En ellas se empleó la función **tempfile.mkdtemp()**

para crear una carpeta temporal con el prefijo “GenAgans” seguido de un código aleatorio. Esto evita que se sobrescriba la carpeta anterior si la aplicación se lanzó previamente. Luego, se empleó la función **Path()** para definir una variable que almacena la ruta a la carpeta temporal creada. Finalmente, se empleó la función **os.chdir()** que cambia el directorio actual de la terminal de Linux a esta nueva carpeta (Fig. 5).

```
831     dir_temp = tempfile.mkdtemp(prefix="GenAgans")
832     ruta_archivos = f"{Path(dir_temp)}"
833     os.chdir(ruta_archivos)
```

Figura 5. Código para la creación de la carpeta temporal, definición de una variable con su ruta y cambio del directorio actual a la carpeta temporal. Fuente: Elaboración propia.

3.3.2. Estructura y funciones recurrentes del servidor

El servidor comprende el conjunto de funciones que reaccionan a las acciones del usuario. Así, el servidor se definió como “**def server(input, output, session):**” (Fig. 6) y dentro de esta función se incluyeron todas las funciones reactivas, es decir, aquellas que responden a los cambios producidos en la UI y la actualizan en tiempo real.

```
836 ##### Definición del servidor: todos los efectos reactivos
837 def server(input, output, session):
838
839 ##### Variables globales de definición reactiva #####
840 ##### Inicio - Nombres archivos #####
841     nombre_archivo1_completo = reactive.Value("")
842     nombre_archivo2_completo = reactive.Value("")
843     nombre_archivo3_completo = reactive.Value("")
844     nombre_archivo1_sin_extensiones = reactive.Value("")
845     nombre_archivo2_sin_extensiones = reactive.Value("")
846
847 ##### 1. Análisis calidad #####
848     comando1_a_actual = reactive.Value("")
849     comando1_actual = reactive.Value("")
850     analisis_calidad_completado = reactive.Value(False)
851     resultado_html1_calidad = reactive.Value("Todavía no hay resultados")
852     resultado_html2_calidad = reactive.Value("Todavía no hay resultados")
853     comando1_ejecutado = reactive.Value("")
854     comando1_ejecutado = reactive.Value("")
```

Figura 6. Código de definición del servidor y de las variables globales reactivas para el apartado 1 y 2. Fuente: Elaboración propia.

La estructura del código del servidor se dividió en tres partes:

En primer lugar, se definieron una serie de variables globales reactivas. Estas son variables cuyo contenido se actualiza en función de las acciones del usuario. Para definirlas se empleó la función **reactive.Value()**. Esta función permite que su contenido se redefina a lo largo del código del servidor al emplear el método **.set()**. Asimismo, su contenido puede ser accedido mediante el método **.get()**. Esta parte del código está comprendida entre la línea

839 y la 1019. En la [Figura 6](#), se muestra un ejemplo de las variables globales reactivas para la pestaña “Inicio” y “1. Control de calidad de las lecturas”.

En segundo lugar, se definieron de forma reactiva los [comandos de Bash del flujo de trabajo](#). En estos se emplea el método .set() para actualizar el contenido de las variables globales reactivas de tipo “comando_actual”. En la línea de código 848 y 849 de la [Figura 6](#) se muestra un ejemplo de este tipo de variables globales reactivas. Mediante el método .get() se obtiene la información contenida en las variables reactivas para personalizar los comandos. Esta parte del código está comprendida entre la línea 1022 y la 1223. En la [Figura 7](#) se muestra un ejemplo para los comandos de la pestaña 1.

```
1022 ##### Actualización reactiva de los comandos #####
1023 ##### 1. Limpieza Lecturas #####
1024 @reactive.effect
1025 def comando1_a_actualizacion():
1026     comando1_a_actual.set("mkdir 1.Analisis_calidad")
1027
1028 @reactive.effect
1029 def comando1_actualizacion():
1030     comando1_actual.set(f"fastqc -o 1.Analisis_calidad/ {nombre_archivo1_completo.get()} {nombre_archivo2_completo.get()}")
```

[Figura 7.](#) Código para la definición reactiva de los comandos del apartado 1. Fuente: Elaboración propia.

En tercer lugar, se definieron las [funciones que controlan los procesos del servidor](#) e interactúan con las variables de UI. Esta parte del código está comprendida entre la línea 1225 y la 2878. Cada tipo de componente de la UI requiere de unas funciones concretas, por lo que se dan ejemplos y se describen específicamente en los siguientes subapartados. Sin embargo, los decoradores son elementos recurrentes en esta parte del código, por lo que se definen a continuación.

Los decoradores son funciones que envuelven a las funciones del servidor y les permiten reaccionar a la UI automáticamente o cuando se produce un evento concreto. Para la creación de este programa se emplearon los siguientes decoradores:

@render.text: vincula la función definida bajo ella con una caja de texto en la UI y actualiza el texto mostrado de forma reactiva. Se empleó para mostrar textos en las cajas de textos fijos y reactivos.

@reactive.effect: no se vincula con la UI directamente si no que permite actualizar los valores de las variables del servidor de forma automática.

@reactive.event(): permite ejecutar la función bajo ella solo cuando ocurre un evento concreto. Se empleó para ejecutar funciones al presionar botones.

@render.ui: vincula la función definida bajo ella con una caja en la UI y permite renderizar diferentes tipos de archivos. Se empleó para mostrar en las cajas de resultados los archivos txt y HTML generados tras lanzar los comandos del flujo de trabajo.

@render.download: permite descargar archivos. Se empleó para gestionar la descarga de archivos al presionar el botón de descarga de la UI.

3.3.3. Estructura y funciones recurrentes de la UI

La UI de la aplicación se dividió en distintas pestañas. Cada una de las pestañas corresponde a una etapa distinta del flujo de trabajo exceptuando el apartado “Inicio” y “Resumen” (Fig 8a). Todas las pestañas siguen un patrón de estructura similar en el que se insertaron diferentes tipos de cajas (Fig. 8b, c y d) y botones (Fig. 8e) en función de las necesidades de cada apartado.

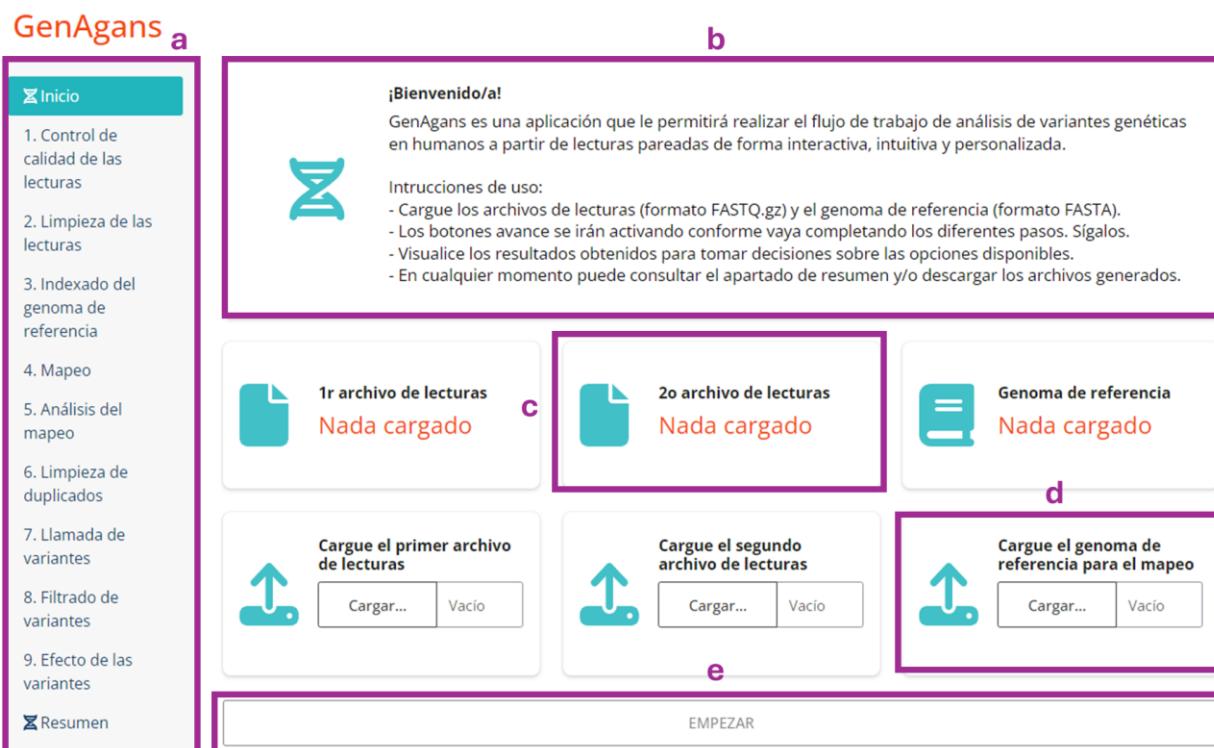


Figura 8. Interfaz de la pestaña “Inicio” de GenAgans previa carga de archivos. En morado se resaltan diferentes partes de la UI: a) Contenedor de pestañas; b) Caja de texto fijo; c) Caja de texto reactivo; d) Caja de carga de archivos; e) Botón de cambio de pestaña. Fuente: Elaboración propia.

La estructura general de la UI (también llamado *layout*) se definió con la función **ui.page_fluid()** (Fig. 9). Esta función crea un contenedor de página fluido, es decir, una interfaz en la que los componentes de la UI insertados se adaptan al tamaño de la pantalla en la que se muestran. Dentro de ui.page_fluid() se insertaron las siguientes funciones:

ui.head_content(): permite agregar metadatos al HTML de la UI. En este caso se añadieron un archivo de CSS con ajustes estéticos, y un archivo CSS y un archivo de JavaScript para ajustar la visualización del HTML resultante de la ejecución del comando para la limpieza de las lecturas. Para la adición de los archivos CSS se empleó la función **ui.include_css()** y para los de JavaScript **ui.include_js()**.

ui.panel_title(): agrega un título a la interfaz. En este caso “GenAgans”.

ui.navset_pill_list(): agrega un contenedor de pestañas a la interfaz (Fig. 8a). Dentro de este se definieron las diferentes pestañas mediante la función **ui.nav_panel()** donde se especifica el nombre que aparece en la interfaz para cada pestaña y el nombre de la variable con la que se identificará cada pestaña en el servidor. Al final se especifica la anchura que debe tener el contenedor de pestañas respecto a su contenido con el parámetro “widths” y la variable con la que se identifica el conjunto de pestañas en el servidor con el parámetro “id”.

```

806 app_ui = ui.page_fluid(
807   ui.head_content(ui.include_css("./custom_css/custom.css"),
808   ui.include_css("./comando_css_html2/comando_css.css"),
809   ui.include_js("./comandos_java_html2/comandos_java.js")
810   ),
811   ui.panel_title("GenAgans", "GenAgans"),
812   ui.navset_pill_list(
813     ui.nav_panel("Inicio", Inicio, icon=icon_svg("dna")),
814     ui.nav_panel("1. Control de calidad de las lecturas", Control_calidad),
815     ui.nav_panel("2. Limpieza de las lecturas", Limpieza_lecturas),
816     ui.nav_panel("3. Indexado del genoma de referencia", Indexado),
817     ui.nav_panel("4. Mapeo", Mapeo),
818     ui.nav_panel("5. Análisis del mapeo", Analisis_mapeo),
819     ui.nav_panel("6. Limpieza de duplicados", Limpieza_duplicados),
820     ui.nav_panel("7. Llamada de variantes", Llamada_variantes),
821     ui.nav_panel("8. Filtrado de variantes", Filtrado_variantes),
822     ui.nav_panel("9. Efecto de las variantes", Efecto_variantes),
823     ui.nav_panel("Resumen", Resumen, icon=icon_svg("dna")),
824     widths=(2,10),
825     id="panel",
826   ),
827 )

```

Figura 9. Código para la definición de la estructura general de la UI y las distintas pestañas que la componen.

Fuente: Elaboración propia.

Tras definir la estructura general de la UI y las pestañas que la componen, se definió la estructura de la UI para cada una de ellas. Para ello, cada variable asociada a una pestaña fue definida con ui.page_fluid().

Dentro del ui.page_fluid() se insertó la función ui.layout_column_wrap(). Esta función permite agrupar diferentes componentes de la UI en columnas dentro de una misma fila. Por ejemplo, en la Figura 8, la caja de texto fijo, las tres cajas de texto reactivo, las tres cajas de carga de archivos y el botón de cambio de pestaña, se definieron cada una de ellas dentro de su propia ui.layout_column_wrap(). Por tanto, se definieron 4 ui.layout_column_wrap(), una para cada fila de componentes de la pestaña “Inicio”. En el Anexo 3 se muestra la definición completa de la UI para dicho apartado.

En la Figura 10 se muestra un ejemplo del uso de estas funciones para la definición de las dos primeras filas de la pestaña de “Inicio”. Por lo que se observa un solo ui.page_fluid() y dos ui.layout_column_wrap().

```
15 Inicio = ui.page_fluid(  
16   ui.layout_column_wrap(  
17     ui.value_box(  
18       "¡Bienvenido/a!",  
19       ui.output_ui("texto_introduccion"),  
20       showcase=icon_svg("dna"),  
21     ),  
22   ),  
23   ui.layout_column_wrap(  
24     ui.value_box(  
25       "1r archivo de lecturas",  
26       ui.output_text("archivo1_nombre"),  
27       height="150px",  
28       showcase=icon_svg("file"),  
29     ),  
30     ui.value_box(  
31       "2o archivo de lecturas",  
32       ui.output_text("archivo2_nombre"),  
33       height="150px",  
34       showcase=icon_svg("file"),  
35     ),  
36     ui.value_box(  
37       "Genoma de referencia",  
38       ui.output_text("archivo3_nombre"),  
39       height="150px",  
40       showcase=icon_svg("book"),  
41     ),  
42   ),  
),
```

Figura 10. Recorte del código para la definición de la UI de la pestaña “Inicio”. Se observan la función para la definición de la estructura de la página, las funciones para la estructuración de esta en filas y las funciones para la caja de texto fijo y las cajas de texto reactivo. Fuente: Elaboración propia.

Dentro de cada fila definida por ui.layout_column_wrap() las cajas se insertan de mediante ui.value_box(). Esta función permite mostrar datos clave dentro de una caja. Se empleó para insertar cajas de texto fijo, texto reactivo y cajas de renderizado de resultados. Por ejemplo, en la Figura 10, se observa su uso para definir la caja de texto fijo (Fig. 8b) y su uso por triplicado para definir las cajas de texto reactivo (Fig. 8c).

Como se observa en el ejemplo de la Figura 10, cada caja o botón insertado dentro de cada fila definida por ui.layout_column_wrap() se define de distinta forma dependiendo de su utilidad. A continuación, se describen las funciones empleadas para definir cada una de estas cajas o botones.

3.3.1. Cajas de textos fijos

Para definir las cajas de textos fijos, en la UI se empleó la función ui.output_ui() dentro de un ui.value_box() (Fig. 10). Esta función permite añadir información a un contendor de la interfaz, en este caso se empleó para mostrar por pantalla texto en formato HTML (Fig. 8b).

La variable definida dentro de ui.output_ui(), en este caso “texto_introduccion”, se emplea en la función del servidor para introducir el texto que se muestra por pantalla. En primer lugar, se empleó el decorador @render.text para mostrar por pantalla el texto indicado en la función. Luego, se definió la función con el nombre de la variable indicada en la UI (“texto_introduccion”) y en la salida de la función se empleó la función ui.HTML() para indicar el texto a mostrar por pantalla en formato HTML (Fig. 11).

Este conjunto de funciones se empleó para la caja de texto de la introducción, la caja de texto del resumen, las cajas de texto de los apartados opcionales y la caja de advertencia del apartado 6.

```

1230 @render.text
1231 def texto_introduccion():
1232     return ui.HTML('GenAgans es una aplicación que le permitirá realizar el flujo de trabajo de análisis de variantes genéticas \
1233     en humanos a partir de lecturas pareadas de forma interactiva, intuitiva y personalizada. <br> \
1234     <br>\n    Intrucciones de uso: <br> \
1235     - Cargue los archivos de lecturas (formato FASTQ.gz) y el genoma de referencia (formato FASTA). <br> \
1236     - Los botones avance se irán activando conforme vaya completando los diferentes pasos. Sígalos. <br> \
1237     - Visualice los resultados obtenidos para tomar decisiones sobre las opciones disponibles. <br> \
1238     - En cualquier momento puede consultar el apartado de resumen y/o descargar los archivos generados.')
1239

```

Figura 11. Código del servidor para la definición del texto fijo a mostrar en la introducción. Se observa el decorador @render.text, la función definida con el nombre de la variable indicada en la UI y la función ui.HTML() que contiene el texto a mostrar. Fuente: Elaboración propia.

3.3.2. Cajas de carga de archivos

Para definir las cajas de carga de archivos en la UI se empleó la función `ui.input_file()` dentro de `ui.value_box()`. Esta función crea un campo para la carga de archivos. En él se especifica el nombre que recibe la variable que contiene el archivo (por ejemplo “archivo1”) y el texto que se quiere mostrar por pantalla (este campo se dejó en blanco). Seguido de una serie de argumentos: “multiple=False” impide cargar más de un archivo en ese campo, “accept=” permite introducir una lista de extensiones aceptadas, “button_label=” especifica la palabra a mostrar sobre el botón de carga y “placeholder=” especifica la palabra que aparece al lado del botón de carga mientras no se haya cargado ningún archivo.

Esta función se empleó únicamente en el apartado “Inicio” para definir las tres cajas de carga de archivo (Fig. 8d). En la Figura 12 se muestra su uso en el código de la UI.

```

43     ui.layout_column_wrap(
44         ui.value_box(
45             "Cargue el primer archivo de lecturas",
46             ui.input_file("archivo1", "", multiple=False, accept=".fastq.gz", ".fq.gz"),
47             button_label="Cargar...", placeholder="Vacio",
48             height="150px",
49             showcase=icon_svg("upload"),
50         ),
51         ui.value_box(
52             "Cargue el segundo archivo de lecturas",
53             ui.input_file("archivo2", "", multiple=False, accept=".fastq.gz", ".fq.gz"),
54             button_label="Cargar...", placeholder="Vacio",
55             showcase=icon_svg("upload"),
56         ),
57         ui.value_box(
58             "Cargue el genoma de referencia para el mapeo",
59             ui.input_file("archivo3", "", multiple=False, accept=".fasta", ".fa"),
60             button_label="Cargar...", placeholder="Vacio",
61             showcase=icon_svg("upload"),
62         ),
63     ),

```

Figura 12. Código para la definición de las cajas de carga de archivos del apartado “Inicio”. Fuente: Elaboración propia.

Las variables creadas con `ui.input_file()` no requieren directamente de un código en el servidor que produzca reactividad. Si no que, al cargar un archivo, estas variables almacenan la información del archivo subido. Se puede llamar a dicha información mediante el método `input.` y el nombre de la variable. Por ejemplo, “`input.archivo1()`” contiene el nombre, el peso (en bytes), el tipo de archivo y la ruta del archivo cargado (Fig. 13).

```
[{'name': 'SRR14695036_1.fastq.gz', 'size': 805691927, 'type': 'application/x-gzip', 'datapath': '/tmp/fileupload-y7wj9t2k/tmps9vwue8h/0.gz'}]
```

Figura 13. Ejemplo del contenido de la variable “input.archivo1()”. Fuente: Elaboración propia.

Estos datos se almacenan como una lista de diccionarios. Por tanto, se puede acceder a los datos específicos mediante el uso de corchetes. Un primer par de corchetes para indicar el elemento (diccionario) de la lista a extraer y un segundo par de corchetes para seleccionar, en el diccionario, la clave cuyo valor asociado se quiere extraer. Por ejemplo, “input.archivo1()[0][“datapath”]” extrae la ruta del archivo 1, en el caso de la Figura 13 “tmp/fileupload-y7wj9t2k/tmps9vwue8h/0.gz”. Por otra parte, “input.archivo1()[0][“name”]” extrae su nombre, en este caso “SRR14695036_1.fastq.gz”. En GenAgans, la lista siempre contiene un solo elemento (diccionario) porque la opción de cargar múltiples archivos en un mismo campo de carga se desactivó para todos ellos.

3.3.3. Cajas de texto reactivo

Para definir las cajas de texto reactivo, en la UI se empleó la función **ui.output_text()** dentro de **ui.value_box()**. Esta función permite añadir texto a la caja de la UI de forma reactiva. En ella se especifica el nombre de la variable de la que se tomará el texto a mostrar. En la Figura 10 se observa un ejemplo de su uso para cada una de las 3 cajas de texto reactivo del apartado “Inicio” (Fig. 8c).

En el servidor, se define la función con el nombre asignado a la variable en la UI bajo el decorador **@render.text**. Por ejemplo, para la primera caja de texto reactivo del apartado “Inicio”, la variable se llama “archivo1_nombre” (Fig. 10). Por tanto, el nombre de la función en el servidor que determina su contenido es “archivo1_nombre()” (Fig. 14).

```
1242     @render.text
1243     def archivo1_nombre():
1244         if input.archivo1() is None:
1245             return "Nada cargado"
1246         else:
1247             ruta_archivo1 = input.archivo1()[0]["datapath"]
1248             nombre_archivo1 = input.archivo1()[0]["name"]
1249             reposicion_archivo1 = f"mv {ruta_archivo1} {ruta_archivos}/{nombre_archivo1}"
1250             subprocess.run(reposición_archivo1, shell=True)
1251             return f"{nombre_archivo1}"
```

Figura 14. Código en el servidor para la función de carga de archivos del archivo 1. Fuente: Elaboración propia.

Posteriormente se definió la lógica de la reactividad del texto en el servidor (Fig. 14). Si el archivo 1 no se ha cargado, entonces la función reporta el texto “Nada cargado”. Si sí que

se ha cargado, se define una variable que contiene la ruta al archivo 1, una variable que contiene el nombre del archivo 1 y una variable que contiene el comando para mover el archivo 1 de la ruta en la que se ha cargado a la carpeta temporal de GenAgans. Luego, con la función **subprocess.run()** se lanza el comando a la terminal de Linux para ejecutarlo. Por último, la función devuelve el nombre del archivo cargado a la caja de texto reactivo.

Este conjunto de funciones se empleó para las tres cajas de texto reactivo del apartado “Inicio” (Fig. 8c). También se usó una variante de este conjunto de funciones para las cajas de comandos de las diferentes pestañas. En la Figura 15a se muestra un ejemplo para el comando del apartado “1. Control de calidad de las lecturas”.

GenAgans



Figura 15. Interfaz de la pestaña 1 de GenAgans tras cargar los archivos y ejecutar el comando. En morado se resaltan diferentes partes de la UI: a) Caja de texto reactivo para el comando; b) Botón de inicio de proceso; c) Caja de renderizado de resultados; d) Botón de descarga de archivos; e) Botón de avance entre pestañas.
Fuente: Elaboración propia.

En estos casos el texto mostrado depende de si se cargaron los archivos, en cuyo caso se muestra el comando a ejecutar, o si no se cargaron, en cuyo caso se muestra el mensaje “Cargue los archivos, por favor.”. En la Figura 16 se muestra un ejemplo, en este caso para la caja de texto del comando del apartado “1. Control de calidad de las lecturas”.

```
1310 @render.text
1311 def comando1():
1312     if input.archivo1() is not None and input.archivo2() is not None and input.archivo3() is not None:
1313         return comando1_actual.get()
1314     else:
1315         return "Cargue los archivos, por favor."
```

Figura 16. Código en el servidor para la caja de texto del comando del apartado 1. Fuente: Elaboración propia.

3.3.4. Botones de avance entre pestañas

Para definir los botones de avance entre pestañas, en la UI se empleó la función `ui.input_action_button()` directamente dentro de `ui.layout_column_wrap()`. Esta función inserta un botón en la UI. En ella se especifica el nombre de la variable que lo identificará en el código, el nombre que se muestra en la UI y con el parámetro “disabled=” se indica si el botón debe aparecer activado (“False”) o desactivado (“True”) por defecto. En la [Figura 17](#) se muestra un ejemplo para el botón EMPEZAR del apartado “Inicio” ([Fig. 8e](#)).

```
64     ui.layout_column_wrap(  
65         ui.input_action_button("boton_empezar", "EMPEZAR", disabled=True),  
66     ),
```

Figura 17. Código en la UI para la definición del botón “EMPEZAR” del apartado “Inicio”. Fuente: Elaboración propia.

En GenAgans, los botones de avance entre pestañas se mantienen desactivados hasta que se cumplen una serie de requisitos. Esto permite al usuario conocer cuándo aún debe realizar acciones en la pestaña en la que se encuentra antes de proceder al siguiente paso.

Para modular la activación y desactivación de los botones, se modifica el parámetro “disabled=” mediante la función `ui.update_action_button()`. Este debe pasar a “False” cuando se cumple un requisito. Por tanto, el código en el servidor se decoró con `@reactive.effect` para detectar y responder a los cambios de forma automática. Dentro de la función decorada se indicó la lógica que decide cuándo se mantiene desactivado (“True”) el botón y cuándo se activa (“False”).

En la [Figura 18](#) se observa el ejemplo del código en el servidor para el botón “EMPEZAR” del apartado “Inicio” ([Fig. 8e](#)). Si no se han cargado los tres archivos necesarios para el flujo de trabajo, el botón de empezar se mantiene desactivado. Si se han cargado, ya se puede iniciar el flujo de trabajo. Por tanto, se activa el botón “EMPEZAR” (cambio de pestaña) y el botón “Iniciar análisis de calidad de las lecturas” (inicio de proceso) correspondiente al primer apartado del flujo de trabajo ([Fig. 15e](#)). Además, esto actualiza automáticamente las variables globales reactivas para los nombres de los tres archivos cargados, con y sin extensiones, pues serán empleados en otras funciones del flujo de trabajo.

```

1278     @reactive.effect
1279     def botones_inicio_procesos01():
1280         if input.archivo1() is not None and input.archivo2() is not None and input.archivo3() is not None:
1281             ui.update_action_button("boton_empezar", disabled=False)
1282             ui.update_action_button("boton_analisis_calidad", disabled=False)
1283             extensiones = [".fastq.gz", ".fq.gz"]
1284             nombre_archivo1 = input.archivo1()[0]["name"]
1285             nombre_archivo2 = input.archivo2()[0]["name"]
1286             nombre_archivo3 = input.archivo3()[0]["name"]
1287             archivo1_sin_extensiones = nombre_archivo1
1288             archivo2_sin_extensiones = nombre_archivo2
1289             for i in extensiones:
1290                 archivo1_sin_extensiones = archivo1_sin_extensiones.replace(i, '')
1291                 archivo2_sin_extensiones = archivo2_sin_extensiones.replace(i, '')
1292                 nombre_archivo1_completo.set(nombre_archivo1)
1293                 nombre_archivo2_completo.set(nombre_archivo2)
1294                 nombre_archivo3_completo.set(nombre_archivo3)
1295                 nombre_archivo1_sin_extensiones.set(archivo1_sin_extensiones)
1296                 nombre_archivo2_sin_extensiones.set(archivo2_sin_extensiones)
1297             else:
1298                 ui.update_action_button("boton_empezar", disabled=True)

```

Figura 18. Código en el servidor para la definición del botón “EMPEZAR” del apartado “Inicio”. Fuente: Elaboración propia.

El código de la Figura 18 se replicó de forma simplificada para el resto de las pestañas. En la Figura 19 se muestra el ejemplo del código para la actualización del botón del apartado 1 (Fig. 15e). Este código simplemente verifica si el proceso del apartado se ha completado y habilita el botón para pasar el siguiente.

```

1365     @reactive.effect
1366     def siguiente1():
1367         if analisis_calidad_completado.get() is True:
1368             ui.update_action_button("boton_siguiente1", disabled=False)

```

Figura 19. Código en el servidor para la actualización del botón “siguiente” del apartado 1. Fuente: Elaboración propia.

Para que se produzca el cambio entre pestañas, el código del servidor debe reaccionar al pulsado del botón. Para ello el código se decora tanto con @reactive.effect como con @reactive.event(). Dentro de @reactive.event() se indica input. + el nombre de la variable del botón. Esta variable contiene un número que empieza en 0 y aumenta en 1 cada vez que se aprieta el botón. Cuando se detecta un cambio en dicho número, se acciona la función de debajo del decorador @reactive.event(). Dentro de esta se empleó la función **ui.update_navs()** que produce el cambio de pestaña en la UI. En ella se especifica el identificador del conjunto de pestañas (“panel”) y con el parámetro “selected=” se indica la pestaña a la que se cambiará al pulsar el botón.

En la [Figura 20](#) se muestra el ejemplo del código del servidor empleado para accionar el cambio de la pestaña “Inicio” a la pestaña “1. Control de calidad de las lecturas”. Este código se replicó para cada una de las pestañas excepto “Resumen”.

```
1301 |     @reactive.effect
1302 |     @reactive.event(input.boton_empezar)
1303 |     def inicio_a_1():
1304 |         ui.update_navs("panel", selected="1. Control de calidad de las lecturas")
```

Figura 20. Código en el servidor para producir el cambio de la pestaña de “Inicio” a la pestaña del apartado 1.

Fuente: Elaboración propia.

3.3.5. Paneles de opciones para la personalización de los comandos del flujo de trabajo

Algunos comandos del flujo de trabajo disponen de diversas opciones que permiten su personalización. En esta aplicación se programaron algunas de estas opciones para que el usuario pueda hacer uso de ellas sin tener que indagar en su documentación específica.

Para definir la estructura de los paneles de opciones en la UI se empleó la función `ui.layout_columns()` en lugar de `ui.layout_column_wrap()`. La diferencia radica en que `ui.layout_column_wrap()` permite la reposición de los elementos que contiene en una fila adicional cuando no caben en pantalla, evitando que el ancho de estos se modifique. Sin embargo, `ui.layout_columns()` permite una estructura más rígida, donde se prioriza mantener los elementos en una misma fila, a costa de modificar su anchura para adaptarse a la pantalla. En el [Anexo 4](#) se observa una comparación entre las consecuencias del uso de `ui.layout_columns()` y `ui.layout_column_wrap()`, concretamente en el caso de las opciones para el comando de adición o reemplazo de RG.

La elección de `ui.layout_columns()` para los paneles de opciones se basó en que en estos casos no se emplearon cajas rígidas como las `ui.value_box()` si no que se usaron tarjetas, cuya anchura no resulta esencial. Para definir las tarjetas se empleó la función `ui.card()`. Esta crea un contenedor en el que se pueden insertar diferentes elementos que permiten la introducción de datos por parte del usuario. En la UI esto se observa como un borde que recoge distintos elementos ([Anexo 4](#)).

Dentro de las `ui.card()` se insertaron diversas funciones de tipo “input”. Las empleadas para la creación de GenAgans fueron:

ui.input_text(): crea una caja en la que se puede introducir texto. Dentro de esta función se especifica el nombre de la variable que la identifica y que contendrá el texto escrito en la caja. Luego, se especifica el texto que se muestra por pantalla y el texto que aparece por defecto en la caja.

ui.input_numeric(): crea una caja en la que se puede introducir un número. Dentro de esta función se especifica el nombre de la variable que la identifica y que contendrá el número escrito en la caja. Luego, se especifica el texto que se muestra por pantalla, el número que aparece por defecto en la caja y el número mínimo y/o máximo aceptado.

ui.input_checkbox(): crea una casilla que se puede marcar y desmarcar. Dentro de esta función se especifica el nombre de la variable que la identifica. Esta es de tipo booleano, es decir, su valor será “True” si la casilla está marcada o “False” si está desmarcada. Luego se especifica el texto que se muestra por pantalla y el valor por defecto de la variable. Esta función se empleó generalmente para anidar otras funciones como ui.input_numeric() e ui.input_text(). En el [Anexo 5](#) se muestra un ejemplo del uso de estas funciones en el código UI para el apartado “2. Limpieza de las lecturas”. En el [Anexo 6](#) se muestra la apariencia del panel de opciones creado con dichas funciones.

ui.input_select(): crea una caja desplegable de selección de opciones. Dentro de esta función se especifica el nombre de la variable que la identifica. Luego, el texto a mostrar por pantalla y un diccionario con el texto a mostrar en el desplegable (clave del diccionario) y el valor que se almacenará en la variable (valor asociado a la clave en el diccionario). Por último, se especifica la opción seleccionada por defecto con el parámetro “selected=”.

ui.input_selectize(): crea una caja en la que se pueden seleccionar y deseleccionar varias opciones. Dentro de esta función se especifica el nombre de la variable que la identifica. Luego, el texto a mostrar por pantalla y una tupla que contiene todas las opciones seleccionables. Por último, se indica el parámetro “multiple=True” para permitir la selección de múltiples opciones al mismo tiempo.

ui.tooltip(): crea una tarjeta con texto que aparece al pasar el ratón por encima de un texto. Esta función solo se empleó para anidar algunas funciones con el objetivo de mostrar por pantalla un texto explicativo de dicha función al pasar el ratón por encima. Por tanto, dentro

de esta función se especifica otra como “ui.input_checkbox()”. Luego, se especifica el texto a mostrar en el mensaje, el identificador de la función y con el parámetro “placement=” a qué lado debe aparecer el texto respecto al ratón.

Las tres últimas funciones descritas solo se emplearon en el código UI para el apartado “9. Efecto de las variantes”. En el [Anexo 7](#) se muestra un ejemplo de su uso en dicho apartado. En el [Anexo 8](#) se muestra la apariencia que tiene en la aplicación al seleccionar algunos de los campos de ui.input_selectize() y al pasar el ratón por una opción anidada bajo ui.tooltip().

En el código del servidor, se accede al contenido de las variables definidas en estas funciones mediante el método “input.” + variable. Esto permite actualizar las variables globales reactivas en función de las acciones del usuario. En la [Figura 21](#) se muestra un ejemplo en el que diversas variables globales se actualizan o no en función de si se han marcado las casillas pertinentes del apartado “9. Efecto de las variantes”. Por defecto, las variables globales para las opciones permanecen vacías (“”). Sin embargo, en caso de que se haya marcado la casilla correspondiente, el valor de la variable global se sustituye con la opción seleccionada. Por ejemplo, “--af” o “--fields”. Además, en el caso de “--fields”, también se define la función global que contendrá los valores seleccionados en la caja de selección múltiple. Por último, si en la caja de texto de la opción adicional se ha escrito algo, también se definirá la función global correspondiente con lo escrito.

```

2396     @reactive.effect
2397     def af_comando():
2398         if input.check_af() is True:
2399             af.set("--af")
2400         else:
2401             af.set("")
2402
2403     @reactive.effect
2404     def fields_comando():
2405         if input.check_fields() is True:
2406             fields.set("--fields")
2407             fields_valor.set(f'"{","".join(input.campos_seleccionados())}"')
2408         else:
2409             fields.set("")
2410             fields_valor.set("")
2411
2412     @reactive.effect
2413     def opcion_adicional9_comando():
2414         if input.opcion_adicional9() is not None:
2415             return opcion_adicional_9.set(input.opcion_adicional9())

```

Figura 21. Código del servidor para la actualización de las variables globales de definición reactiva según las opciones que marque el usuario. Fuente: Elaboración propia.

Luego, el contenido de estas variables globales se emplea en la actualización reactiva de los comandos, por lo que se personalizan con las opciones seleccionadas. En la Figura 22 se muestra el código para la definición reactiva del comando de predicción de variantes.

```

1204 |     @reactive.effect
1205 |     def comando9_actualizacion():
1206 |         comando9_actual.set(f"vep -i ./8.Filtrado_variantes/variantes.vcf \
1207 |                         -o ./9.Efecto_variantes/efecto_variantes.tsv \
1208 |                         --database --species homo_sapiens --force_overwrite --tab \
1209 |                         {assembly.get()} {assembly_valor.get()} \
1210 |                         {clin_sig_allele.get()} {clin_sig_allele_valor.get()} \
1211 |                         {sift.get()} {sift_valor.get()} \
1212 |                         {polyphen.get()} {polyphen_valor.get()} \
1213 |                         {gene_phenotype.get()} \
1214 |                         {variant_class.get()} \
1215 |                         {symbol.get()} \
1216 |                         {protein.get()} \
1217 |                         {hgvs.get()} \
1218 |                         {spdi.get()} \
1219 |                         {check_existing.get()} \
1220 |                         {uniprot.get()} \
1221 |                         {af.get()} \
1222 |                         {opcion_adicional_9.get()} \
1223 |                         {fields.get()} {fields_valor.get()}")

```

Figura 22. Código para la definición reactiva del comando VEP en función de las opciones que marque el usuario. Fuente: Elaboración propia.

En el Anexo 9 se muestra el comando del apartado “9. Efecto de las variantes” tras seleccionar todas casillas de opciones y, por tanto, actualizar el contenido de todas las variables globales mostradas en la Figura 22. En el Anexo 9 no se han seleccionado todos los campos de la caja de selección múltiple para simplificar la visualización.

3.3.6. Botones de inicio de procesos

Para definir los botones de inicio de procesos en la UI se emplearon las mismas funciones que para los botones de cambio de pestaña: ui.layout_column_wrap() y ui.input_action_button(). En la Figura 23 se muestra un ejemplo del código para el apartado 9. En la Figura 15b se muestra un ejemplo del botón en la UI del apartado 1.

```

551 |     ui.layout_column_wrap(
552 |         ui.input_action_button("boton_VEP", "Iniciar predicción del efecto de las variantes",
553 |                                 disabled=True, icon=icon_svg("circle-play")))

```

Figura 23. Código UI para el botón de inicio de ejecución del comando del apartado 9. Fuente: Elaboración propia.

Sin embargo, el código en el servidor es muy diferente. En este caso la activación del botón tiene que lanzar los comandos almacenados en las variables globales reactivas a la terminal

de Linux para que se procese la información. Para ello, en el código del servidor se emplearon de nuevo los decoradores `@reactive.effect` y `@reactive.event()` para el botón de inicio del proceso. Todas las funciones de inicio de proceso siguen el siguiente esquema: Primero, se actualizan las variables globales de “comando ejecutado” mediante el método `.set()` a partir de la información contenida en las variables globales de “comando actual” mediante el método `.get()`. Las variables de “comando actual” contienen el comando a ejecutar con las opciones marcadas por el usuario, la [Figura 22](#) muestra un ejemplo. Así, cuando se presiona el botón de inicio, las variables “comando ejecutado” registran el comando con las opciones seleccionadas en el momento de la ejecución. Esto es útil para generar un script con los comandos ejecutados en el apartado “Resumen”, independientemente de si el usuario cambia las opciones a posteriori sin llegar a lanzar el comando.

En segundo lugar, mediante la función `ui$Progress()` se muestra por pantalla una barra de carga. El avance de la barra de carga se va actualizando conforme se van completando las ejecuciones de los comandos en la terminal de Linux. Para ello, se emplea el método `“set()”` para ir actualizando el punto en el que se detiene la barra.

En tercer lugar, se definen variables que recogen la información de los comandos almacenados en las variables globales. Después, se emplea la función `subprocess.run()` para lanzar los comandos a la terminal de Linux y que se ejecuten.

En aquellos casos en los que la ejecución de los comandos genera archivos de resultados, se emplea la función `open()` para abrir el archivo. Todas las líneas del archivo se guardan como una lista en una función (generalmente “`raw_html`”). Luego, esas líneas contenidas en la lista se unen para formar una única cadena de texto mediante el método `.join()` y se almacenan en otra variable (generalmente “`cadena`”). De esta forma, este bloque de código permite extraer la información almacenada en un archivo HTML o txt. Por último, el texto extraído se almacena en una variable global mediante el método `.set()`.

Finalmente, se actualiza la variable global de compleción del proceso que pasa a contener un “True”, por ejemplo “`efecto_variantes_completado.set(True)`”. En la [Figura 24](#) se muestra un ejemplo del código para el apartado “9. Efecto de las variantes”.

```
2443     @reactive_effect
2444     @reactive.event(input.boton_VEP)
2445     def efecto_variantes_inicio():
2446         comando9a_ejecutado.set(comando9_a_actual.get())
2447         comando9_ejecutado.set(comando9_actual.get())
2448         with ui.Progress(min=1, max=28) as barra:
2449             barra.set(message="Procesando...", detail="Espere, por favor.")
2450             barra.set(3)
2451             crear_carpeta9 = comando9_a_actual.get()
2452             subprocess.run(crear_carpeta9, shell=True)
2453             barra.set(8)
2454             predecir_efecto_variantes = comando9_actual.get()
2455             subprocess.run(predecir_efecto_variantes, shell=True)
2456             barra.set(13)
2457             with open("./9.Efecto_variantes/efecto_variantes.tsv_summary.html") as archivo:
2458                 raw_html = archivo.readlines()
2459                 cadena = "\n".join(raw_html)
2460                 resultado_html_VEP.set(cadena)
2461                 barra.set(18)
2462                 procesar_tabla_VEP()
2463                 barra.set(23)
2464                 with open("./9.Efecto_variantes/tabla_efecto_variantes.html") as archivo2:
2465                     raw_html2 = archivo2.readlines()
2466                     cadena2 = "\n".join(raw_html2)
2467                     resultado_tabla_VEP.set(cadena2)
2468                     barra.set(28)
2469                     efecto_variantes_completado.set(True)
```

Figura 24. Código en el servidor para la ejecución de los comandos del apartado 9. Fuente: Elaboración propia.

En casos muy concretos, el formato que adoptan los archivos HTML o txt tras pasar por este proceso no es el adecuado para mostrarse correctamente en la aplicación. Por tanto, algunos de ellos requieren la definición de una función específica para completar su formateo. En tal caso, la función se llama dentro del código de inicio de proceso (Fig. 24 línea de código 2462).

Por ejemplo, los datos contenidos en el archivo .tsv que se obtiene tras completar la predicción del efecto de las variantes en el apartado 9, no permite su conversión a tabla correctamente. Esto es debido a que el archivo presenta varias líneas de cabecera. Para sortear este problema se definió la función “procesar_tabla_VEP()”. Esta función no requiere un decorador ya que no reacciona directamente a la UI. En ella se abre el archivo .tsv generado con la función open(). Luego se seleccionan todas las líneas que no empiezan por “##” mediante el método `.startswith()`, ya que son las que corresponden a la cabecera. Mediante el método “`.join()`” se unen las líneas seleccionadas. Luego, se convierten en un dataframe separado por tabulaciones mediante la función `pd.read_csv()` y `StringIO()`. Este dataframe se convierte a formato HTML mediante el método `.to_html()`. Esto facilita su

posterior interpretación por parte de Shiny. Por último, el dataframe convertido en HTML se guarda como archivo mediante la función open() y el método **.write()**. De esta forma, el archivo HTML generado contiene los datos del .tsv generado por VEP en un formato tabular que es leído con facilidad por Shiny y que se representa correctamente en la UI (Fig. 25).

```
2433     def procesar_tabla_VEP():
2434         with open("./9.Efecto_variantes/efecto_variantes.tsv", 'r') as archivotsv:
2435             lineas_no_cabecera = [linea for linea in archivotsv if not linea.startswith('#')]
2436             lineas_tabla = ''.join(lineas_no_cabecera)
2437             dataframe_tabla = pd.read_csv(StringIO(lineas_tabla), sep='\t')
2438             tabla_html = dataframe_tabla.to_html(index=False)
2439             with open("./9.Efecto_variantes/tabla_efecto_variantes.html", "w") as archivohtml:
2440                 archivohtml.write(tabla_html)
```

Figura 25. Código en el servidor para el ajuste de formato de la tabla de resultados de VEP. Fuente: Elaboración propia.

3.3.7. Cajas de renderizado de resultados

Para definir las cajas de resultados en el código UI se empleó ui.value_box() dentro de un ui.layout_column_wrap(). La función específica empleada para definir estas cajas fue ui.output_ui(). De forma similar a las cajas de texto fijo. Sin embargo, en este caso, en todas ellas se definió el parámetro “full_screen=” como “True”. Este parámetro habilita un botón que aparece abajo a la derecha de la caja al pasar el ratón por encima (Fig. 15c). Al pulsarlo abre en pantalla completa el contenido asignado a la variable de ui.output_ui(). Por tanto, permite que la visualización de los archivos esté comprimida en la caja por defecto y que, al pulsar el botón de expandir, se puedan observar a pantalla completa. En el Anexo 10 se muestra la comparativa entre la caja de resultados del apartado 2 contraída y expandida.

En la Figura 26 se muestra un ejemplo del código UI para el apartado 9.

```
563     ui.layout_column_wrap(
564         ui.value_box(
565             "Informe de resultados tabulados de la predicción del efecto de las variantes:",
566             ui.output_ui("tabla_VEP"),
567             showcase=icon_svg("scroll"),
568             full_screen=True,
569         ),
570     ),
```

Figura 26. Código UI para la caja de resultados de la tabla de VEP en el apartado 9. Fuente: Elaboración propia.

En el código del servidor se empleó el decorador @render.ui junto a una función nombrada como el identificador de la variable descrita en la UI, en este ejemplo, “tabla_VEP”. Luego se empleó función ui.HTML() donde se introdujo la variable global que contiene los datos del archivo a mostrar (Fig. 27).

```

2477     @render.ui
2478     def tabla_VEP():
2479         return ui.HTML(resultado_tabla_VEP.get())

```

Figura 27. Código en el servidor para mostrar en la UI el resultado de la tabla de VEP en el apartado 9. Fuente: Elaboración propia.

3.3.8. Botones de descarga de archivos

Los botones de descarga de resultados se definieron en la UI mediante la función ui.download_button() dentro de un ui.layout_column_wrap(). Esta función muestra un botón en la interfaz que permite descargar los archivos contenidos en la variable que se define dentro de ella (Fig. 15d). Para ello, se especifica el nombre de la variable y el texto a mostrar sobre el botón de descarga. En la Figura 28 se muestra un ejemplo para el botón del apartado 9.

```

571     ui.layout_column_wrap(
572         ui.download_button("descarga_resultados9", "Descargar resultados", icon=icon_svg("download")),

```

Figura 28. Código UI para el botón de descarga de resultados del apartado 9. Fuente: Elaboración propia.

Todas las pestañas excepto “Inicio” y “Resumen” tienen un botón de descarga para descargar la carpeta con los archivos generados en su respectivo paso del flujo de trabajo. Por tanto, todos ellos se definieron con el mismo patrón en el código del servidor. En él se emplea el decorador @render.download y una función con el mismo nombre que la variable definida en la UI. Dentro de esta se emplea la función zipfile.ZipFile() para generar un archivo comprimido (.zip). Luego, se define una variable que contiene todos los archivos incluidos en la carpeta del apartado (en el apartado 9 es “9.Efecto_variante”) mediante la función glob.glob(). Mediante un “bucle for” se recorren todos los archivos contenidos en la carpeta y con el método .write() se van añadiendo al archivo comprimido. Finalmente, la función devuelve el archivo .zip que contiene todos los archivos de la carpeta. Así, cuando se pulsa el botón de descarga, se inicia automáticamente su descarga. En la Figura 29 se muestra un ejemplo del código para el apartado 9.

```

2482     @render.download
2483     def descarga_resultados9():
2484         with zipfile.ZipFile("Resultados_prediccion_efecto_variante.zip", "w") as archivozip:
2485             archivos_descarga = glob.glob("./9.Efecto_variante/*")
2486             for archivo in archivos_descarga:
2487                 archivozip.write(archivo)
2488         return "Resultados_prediccion_efecto_variante.zip"

```

Figura 29. Código en el servidor para la descarga de los archivos generados en el apartado 9. Fuente: Elaboración propia.

En el apartado “Resumen” también se programó un botón de descarga de resultados. Esta vez, se comprimen todos los archivos de la carpeta padre del flujo de trabajo. Así, al presionar dicho botón se descargan todas las carpetas y archivos generados hasta el momento. Aunque el código de la UI es el mismo que para el resto de los apartados, en el servidor se ajustó el código para permitir la compresión y descarga de todo. Para ello se empleó la función `os.walk()` que recorre de forma recursiva las carpetas generadas para obtener información sobre ellas y los archivos que contienen. Dado que este proceso es pesado, se añadió una barra de carga similar a la empleada en durante la ejecución de comandos. En el [Anexo 11](#) se muestra la variación del código para el apartado “Resumen”.

3.3.9. Generación de un script personalizado

En el apartado de “Resumen” se incluyó una recopilación de los comandos empleados, los resultados obtenidos, un botón de descarga de todos los archivos y una caja donde se reportan todos los comandos empleados en forma de script acompañada de un par de botones, para descargarlo en formato .sh y .txt. Dada la cantidad de información por pantalla que suponen, estos resúmenes se presentan en la interfaz dentro de desplegables. En el [Anexo 12](#) se muestra la interfaz con el desplegable plegado y desplegado para el script.

Para definir los desplegables en la UI se emplearon dos nuevas funciones: `ui.accordion()` y `ui.accordion_panel()`. La primera crea una un contenedor para los paneles desplegables, mientras que la segunda define el contenido de cada uno de ellos ([Fig. 30](#)).

```
584 |     ui.accordion(  
585 |         ui.accordion_panel("Resumen de los comandos ejecutados:",  
586 |             ui.layout_column_wrap(  
| )
```

Figura 30. Código UI para los desplegables del apartado “Resumen”. Fuente: Elaboración propia.

Para la generación del script personalizado se hizo una recopilación de todas las variables de “comandos ejecutados” y se dispusieron bajo una función decorada con `@reactive.effect` en el código del servidor. Como dichas variables almacenan el comando empleado en cada ejecución con las opciones elegidas por el usuario en ese momento, el script refleja solo aquellos comandos ejecutados independientemente de las opciones modificadas tras ello. Por una parte, se definió una función global con todos estos comandos en formato HTML para mostrar el script en la caja de texto reactivo de la interfaz. Por otra parte, se definió una función global con el mismo contenido en formato .txt/.sh para su descarga ([Anexo 13](#)).

4. Resultados y discusión

La aplicación del conjunto de herramientas, comandos, funciones y métodos descritos resultó en la creación de GenAgans, una aplicación interactiva, intuitiva y personalizable para el análisis de variantes genéticas en humanos.

GenAgans se dividió en 11 apartados o pestañas. Cada una de ellas se especializó en una parte del flujo de trabajo de análisis de variantes genéticas humanas, exceptuando “Inicio” y “Resumen”.

En las [Figuras 31-33](#) se muestra la apariencia de la interfaz de GenAgans para cada uno de los apartados tras completar todo el flujo de trabajo. Por tanto, en la interfaz de observan todos los comandos con las opciones seleccionadas y los informes generados durante el flujo de trabajo contraídos. En dichas figuras también se muestra un esquema del flujo de trabajo en GenAgans y el flujo de archivos que se produce durante este.

Estas figuras se encuentran disponibles en el [repositorio de GitHub](#) en máxima resolución. Además, en el [Anexo 14](#) se disponen figuras ampliadas de la interfaz.

0. Inicio

GenGangs

¡Bienvenido/a!

GenGangs es una aplicación que te permitirá realizar el flujo de trabajo de análisis de variantes genéticas en humanos a partir de lecturas pareadas de forma interactiva, intuitiva y personalizada.

Introducción de uso:

- Carga los archivos de lecturas (formato FASTQ.gz) y el genoma de referencia (formato FASTA).
- Los pasos de avance se irán actualizando conforme vaya considerando los diferentes pasos. Síguilos.
- Visualiza los resultados obtenidos para tomar decisiones sobre las opciones disponibles.
- En cualquier momento puede consultar el apartado de resumen y/o descargar los archivos generados.

1º archivo de lecturas
SRR14695039_1.fa
stq.gz

2º archivo de lecturas
SRR14695039_2.fa
stq.gz

Genoma de referencia
Homo_sapiens.GR
Ch38.dna.primary_assembly.fa

Carga el primer archivo de lecturas

Cargar SRR14695039_1.fastq.gz

Carga el segundo archivo de lecturas

Cargar SRR14695039_2.fastq.gz

Carga el genoma de referencia para el mapa

Cargar Homo_gr

EMPEZAR

2. Limpieza de las lecturas

GenAgans

! Inicio

1 Control de calidad de las lecturas

2 Limpieza de las lecturas

3 Indicador del genoma de referencia

4 Mapeo

5 Análisis del mapeo

6 Limpieza de duplicados

7 Llamada de variantes

8 Filtrado de variantes

9 Efecto de las variantes

Resumen

Este apartado es OPCIONAL, aunque altamente recomendable
Si realmente considera que sus archivos de lecturas no requieren ningún tipo de limpieza puede proceder directamente al siguiente apartado.

Siguiente

Selección de sus preferencias para la limpieza de las lecturas:

Restringir calidad media de las lecturas

Recortar bases en el extremo 5' (inicio) de las lecturas

Recortar bases en el extremo 3' (final) de las lecturas

Restringir la longitud mínima de las lecturas

Eliminar lecturas que tengan una calidad media menor que

30 15 15 50

Número de bases a eliminar al inicio:

Número de bases a eliminar al final:

Eliminar adaptadores

Eliminar secuencias polyG

Eliminar secuencias polyT

Si desea añadir alguna opción adicional al comando, escribáta aquí:

Comando que se ejecutara:

```
fastp -i SRR14695039_1.fastq.gz -l SRR14695039_2.fastq.gz -o
./Lecturas_limpias/SRR14695039_1_clean.fq.gz -O
./Lecturas_limpias/SRR14695039_2_clean.fq.gz --
cut_mean_quality 30 --detect_adapter_for_pe -l 50 -h
./Lecturas_limpias/SRR14695039_1_SRR14695039_2_fastp.h
ml -j
./Lecturas_limpias/SRR14695039_1_SRR14695039_2_fastp.js
```

Iniciar limpieza de las lecturas

fastp report

Summary

Detallado

Descargar resultados

Siguiente

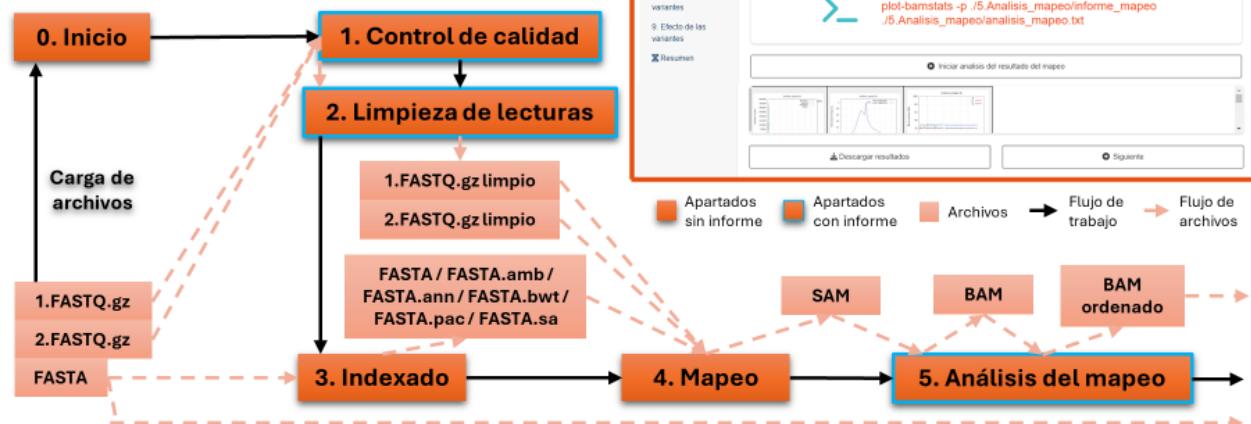


Figura 31. Esquema representativo de la interfaz y el flujo de trabajo de GenAgans desde el apartado “Inicio” hasta el apartado “5. Análisis del mapeo”. Fuente: Elaboración propia.

GenAgans: una aplicación Shiny interactiva e intuitiva para analizar variantes genéticas humanas

0. Inicio: El objetivo principal de este apartado es proporcionar una interfaz en la que cargar los archivos que se emplearán durante el flujo de trabajo. Por tanto, en este apartado se definieron tres cajas de carga de archivos: dos para los archivos de lecturas, que solo aceptan extensiones FASTQ.gz o FQ.gz, y una para el genoma de referencia, que solo acepta extensiones FASTA o FA. Esto se programó para evitar que el usuario pueda subir los archivos equivocados. También se definieron tres las cajas de texto reactivo que indican el nombre de los archivos tras cargarlos, y un pequeño texto con las instrucciones de uso de la aplicación para mejorar la experiencia de usuario ([Fig. 31](#); [Fig. A14.1](#)).

1. Control de calidad de las lecturas: Este apartado permite la ejecución del comando “fastqc” para generar un informe sobre la calidad de las lecturas contenidas en cada uno de los archivos de lecturas cargados. Por tanto, en este apartado se incluyó el comando “fasqc” y se facilitó la visualización de los dos informes en HTML resultantes dentro de la interfaz de GenAgans. La información contenida en estos informes sirve de apoyo para la toma de decisiones en el siguiente apartado ([Fig. 31](#); [Fig. A14.2-3](#)).

2. Limpieza de las lecturas: Aunque este apartado se indica como altamente recomendable en GenAgans, se programó como opcional para no forzar al usuario a realizar la limpieza de sus archivos de lecturas si no lo considera necesario. Así, si el usuario lo considera conveniente, puede proceder directamente al apartado 3. Este apartado permite el filtrado y recorte personalizado de las lecturas analizadas en el apartado anterior mediante el uso del comando “fastp”, y la generación de un informe sobre la calidad de las lecturas tras su limpieza ([Tabla 2](#)). Por tanto, en este apartado, se definieron diversas cajas de opciones adicionales para personalizar el comando, se especificó una caja de texto reactiva con el comando “fastp” que se actualiza al seleccionar y deseleccionar las opciones, y se facilitó la visualización del informe en HTML resultante dentro de la interfaz de GenAgans. Este informe facilita evaluar si la calidad de las lecturas sigue siendo baja y es necesario repetir la limpieza con parámetros más estrictos, o si las lecturas ya están listas para seguir con el flujo de trabajo. En este apartado se generan dos archivos FASTQ.gz con las lecturas limpias ([Fig. 31](#); [Fig. A14.4-6](#)).

3. Indexado del genoma de referencia: Este apartado permite indexar el genoma de referencia cargado. En él se especificó el comando de “bwa index”. Tras su ejecución se generan diversos archivos de indexado del genoma de referencia ([Fig. 31](#); [Fig. A14.7](#)).

4. Mapeo: Este apartado permite el mapeo de las lecturas limpias obtenidas en el apartado 2 contra el genoma de referencia indexado en el apartado 3. Por tanto, en él se especificó el comando “bwa mem”. Tras su ejecución se genera un archivo de mapeo SAM ([Fig. 31](#); [Fig. A14.8](#)).

5. Análisis del mapeo: Este apartado permite la conversión del archivo SAM, obtenido en el apartado 4, en un archivo BAM y su ordenación por coordenadas. Además, se genera un informe de estadísticas sobre el mapeo. En él se especificaron los comandos “samtools view”, “sort” y “stats”, y “plot-bamstats”. Los cuatro comandos se agruparon bajo un mismo botón de inicio. Tras su ejecución se facilitó la visualización del informe en HTML resultante dentro de la interfaz de GenAgans ([Fig. 31](#); [Fig. A14.9-10](#)).

6. Limpieza de duplicados: Este apartado permite marcar los duplicados del archivo BAM, comprobar la presencia de RG en el BAM y, en caso de no aparecer todos, añadirlos o reemplazarlos. En primer lugar, se especificó el comando “picard MarkDuplicates” para marcar los duplicados del archivo BAM ordenado por coordenadas procedente del apartado 5. Posteriormente, se añadió un texto de advertencia sobre cómo proceder respecto a la presencia o ausencia de RG. Seguido de un comando para comprobar los RG presentes en el archivo BAM con los duplicados marcados. Luego se ofreció la posibilidad de añadir o reemplazar los RG mediante el comando “picard AddOrReplaceReadGroups” con opciones para personalizar el contenido de cada RG. Este comando solo se muestra, y su botón de inicio solo se activa, si se han comprobado previamente los RG presentes en el archivo BAM. Se facilitó la visualización por pantalla del informe de “picard MarkDuplicates” y el de comprobación de RG. Así, en este apartado se obtiene finalmente un archivo BAM con los duplicados marcados y con todos los RG necesarios ([Fig. 32](#); [Fig. A14.11-14](#)).

7. Llamada de variantes: Este apartado permite indexar el BAM generado en el apartado 6 y el FASTA del genoma de referencia y, finalmente, hacer la llamada de variantes. Se indicaron los comandos “samtools faidx” e “index”, “freebayes” y “rtg vcfstats”. Al comando “freebayes” se le añadieron opciones adicionales para la personalización de los parámetros de la llamada de variantes ([Tabla 3](#)). Los cuatro comandos se agruparon bajo un mismo botón de inicio. Tras su ejecución se genera un archivo VCF con las variantes identificadas y un informe estadístico sobre estas. Por último, se facilitó la visualización del informe en la interfaz de GenAgans ([Fig. 32](#); [Fig. A14.15-16](#)).

6. Limpieza de duplicados

GenAgans

- Inicio
- 1. Control de calidad de las lecturas
- 2. Limpieza de las lecturas
- 3. Indexado del genoma de referencia
- 4. Mapeo
- 5. Análisis del mapeo
- 6. Limpieza de duplicados**
- 7. Llamada de variantes
- 8. Filtado de variantes
- 9. Efecto de las variantes
- Resumen

Comando que se ejecutará para marcar los duplicados:

```
> picard MarkDuplicates --INPUT
  ./6.Análisis_mapeo/mapeo_sorted.bam --OUTPUT
  ./6.Limpieza_duplicados/mapeo_dedup.bam --METRICS_FILE
  ./6.Limpieza_duplicados/MarkDuplicatesMetrics.txt --
  ASSUME_SORTED True
```

Iniciar marcaje de duplicados

Informes de resultados del marcaje de duplicados:

```
#> htjsdk.samtools.metrics.StringHeader
#> MarkDuplicates --INPUT
#> ./6.Análisis_mapeo/mapeo_sorted.bam --OUTPUT
#> ./6.Limpieza_duplicados/mapeo_dedup.bam --METRICS_FILE
#> ./6.Limpieza_duplicados/MarkDuplicatesMetrics.txt
```

Importante:
Los Read Groups (RG) son metadatos identificativos para las lecturas secuenciadas necesarios para realizar la llamada de variantes.
A continuación se realiza una comprobación de los RG presentes en el archivo de mapeo generado. Los campos esperados son: "ID", "LB", "PL", "PU" y "SM". En función de lo observado en el informe deberá tomar una decisión:
- Si el informe aparece vacío, será necesaria su adición. Para ello emplee el último comando de este apartado.
- Si el informe muestra RG pero no para todos los campos, también será necesaria su adición. Para ello emplee el último comando de este apartado teniendo en cuenta que se reemplazarán los valores actuales.
- Si el informe muestra RG para todos los campos, no será necesaria su adición. Puede ignorar el último comando de este apartado y seguir adelante.

Comando que se ejecutará para comprobar los grupos de lectura (Read Groups) actuales:

```
> samtools view -H ./6.Limpieza_duplicados/mapeo_dedup.bam | grep '^@RG' > ./6.Limpieza_duplicados/ReadGroups.txt
```

Iniciar comprobación de Read Groups

Informe de resultados de la comprobación de los Read Groups:

```
@RG ID:default LB:lib1 PL:ILLUMINA SM:sample1 PU:unit1
```

Si desea añadir Read Groups marque esta casilla, puede usar los valores predefinidos o introducir los propios

RGID - ID del grupo de lecturas (campo ID): <input type="text" value="default"/>	RGLB - Nombre de la librería (campo LB): <input type="text" value="lib1"/>	RGPL - Nombre del secuenciador (campo PL): <input type="text" value="ILLUMINA"/>	RGSM - Nombre de la muestra (campo SM): <input type="text" value="sample1"/>	RGPU - Nombre de la cámara (run) (campo PU): <input type="text" value="unit1"/>
---	---	---	---	--

Comando que se ejecutará para añadir los Read Groups:

```
> picard AddOrReplaceReadGroups -I
  ./6.Limpieza_duplicados/mapeo_dedup_copy.bam -O
  ./6.Limpieza_duplicados/mapeo_dedup.bam -RGID default -
  RGLB lib1 -RGPL ILLUMINA -RGSM sample1 -RGPU unit1
```

Iniciar acción de Read Groups

Descargar resultados Siguiente

7. Llamada de variantes

GenAgans

- Inicio
- 1. Control de calidad de las lecturas
- 2. Limpieza de las lecturas
- 3. Indexado del genoma de referencia
- 4. Mapeo
- 5. Análisis del mapeo
- 6. Limpieza de duplicados**
- 7. Llamada de variantes**
- 8. Filtado de variantes
- 9. Efecto de las variantes
- Resumen

Primer comando que se ejecutará para indexar el genoma de referencia:

```
> samtools faidx
  ./7.Llamada_variantes/Homo_sapiens.GRCh38.dna.primary_assembly.fa
```

Segundo comando que se ejecutará para indexar el archivo de mapeo:

```
> samtools index ./6.Limpieza_duplicados/mapeo_dedup.bam
```

Puede añadir opciones al comando para la llamada de variantes si lo deseas:

<input checked="" type="checkbox"/> Ploidía	<input type="checkbox"/> Fracción mínima	<input type="checkbox"/> Cantidad de lecturas mínima	<input type="checkbox"/> Calidad de mapeo mínima
La ploidía del organismo es de:	La fracción mínima de lecturas que apoyan el alelo alternativo para que se considere como una variante debe ser de:	La cantidad mínima de lecturas que apoyan el alelo alternativo para que se considere como una variante debe ser de:	La calidad mínima de mapeo de una lectura necesaria para considerarla en la llamada variantes debe ser de:
<input type="text" value="2"/>	<input type="text" value="0.2"/>	<input type="text" value="5"/>	<input type="text" value="20"/>

Si desea añadir alguna opción adicional al comando, escribala aquí:

Tercer comando que se ejecutará para hacer la llamada de variantes:

```
> freebayes -ploidy 2 -r
  ./7.Llamada_variantes/Homo_sapiens.GRCh38.dna.primary_assembly.fa ./6.Limpieza_duplicados/mapeo_dedup.bam >
  ./7.Llamada_variantes/variantes.vcf
```

Cuarto comando que se ejecutará para obtener un informe de la llamada de variantes:

```
> bgzip ./7.Llamada_variantes/variantes.vcf >
  ./7.Llamada_variantes/informe_variantes.txt
```

Iniciar llamada de variantes

Informe de resultados de la llamada de variantes:

```
Location : ./7.Llamada_variantes/variantes.vcf
Failed Filters : 0
Passed Filters : 4629726
SNPs : 3747468
```

Descargar resultados Siguiente

Apartados sin informe Apartados con informe Archivos → Flujo de trabajo → Flujo de archivos

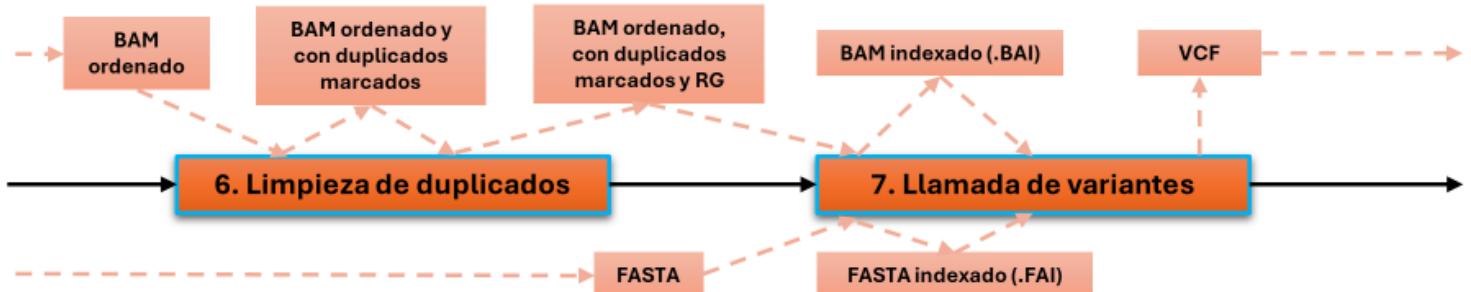


Figura 32. Esquema representativo de la interfaz y el flujo de trabajo de GenAgans para el apartado “6. Limpieza de duplicados” y el apartado “7. Llamada de variantes”. Fuente: Elaboración propia.

8. Filtrado de las variantes: Este apartado permite el filtrado de las variantes identificadas y contenidas en el archivo VCF generado en el apartado 7. Este apartado se programó como opcional para evitar que el usuario se vea forzado a filtrar su archivo de variantes si no lo desea y que, en ese caso, pueda proceder al apartado 9 directamente. En este apartado se especificaron los comandos “vcftools” y “rtg vcfstats”. Al comando “vcftools” se le añadieron opciones adicionales para personalizar el filtrado de las variantes ([Tabla 4](#)). Tras la ejecución de “vcftools” se genera otro VCF con las variantes que han pasado los filtros y con “rtg vcfstats” se genera un informe estadístico sobre estas. Por último, se facilitó la visualización del informe en la interfaz de GenAgans ([Fig. 33; Fig. A14.17-18](#)).

9. Efecto de las variantes: Este apartado permite la anotación de las variantes contenidas en el archivo VCF generado en el apartado 7 u 8, dependiendo de cuál haya sido el último en ejecutarse. Por tanto, en este apartado se indicó el comando “vep” y diversas opciones para personalizarlo ([Tabla 5](#)). Tras su ejecución se generan dos informes, uno con estadísticas generales sobre las variantes anotadas y un informe tabulado con las anotaciones específicas para todas las variantes identificadas. Las columnas de esta tabla varían en función de las opciones añadidas al comando de VEP, por lo que se pueden personalizar. También se ofrece una opción para ordenar y limitar las columnas a incluir en la tabla. Finalmente, se facilitó la visualización del informe general y la tabla de resultados sobre las variantes anotadas con VEP dentro de la propia interfaz de GenAgans ([Fig. 33; Fig. A14.19-25](#)).

10. Resumen: Este apartado permite la visualización de todos los informes obtenidos y todos los comandos ejecutados durante el flujo de trabajo; la descarga de todos los archivos generados; y genera un script personalizado con los comandos ejecutados para permitir la reproducción automatizada del trabajo realizado ([Fig. 33; Fig. A14.26-29](#)).

Con el fin de asegurar que el uso de GenAgans fuera intuitivo, se implementaron algunas medidas de control. Por un lado, si el usuario no carga los archivos, las cajas de texto de “Inicio” indican que no hay nada cargado, las cajas donde se muestran los comandos en el resto de apartados muestran el mensaje “Cargue los archivos, por favor.” y los botones para lanzar dichos comandos se mantienen desactivados. Esto se programó para informar al usuario de que no ha cargado archivos todavía en la sesión y evitar que trate de lanzar los comandos que, sin archivos cargados, fallarían ([Anexo 15](#)).

8. Filtrado de variantes

GenAgans

- X Inicio**
- 1. Control de calidad de las lecturas
- 2. Limpieza de las lecturas
- 3. Indexado del genoma de referencia
- 4. Mapeo
- 5. Análisis del mapeo
- 6. Limpieza de duplicados
- 7. Llamada de variantes
- 8. Filtrado de variantes**
- 9. Efecto de las variantes
- X Resumen**



Este apartado es OPCIONAL
Si no deseas filtrar su archivo de variantes puede proceder directamente al siguiente apartado.

Si desea filtrar las variantes, puede seleccionar alguna de estas opciones o añadir una propia:

Filtrar por calidad mínima

 Eliminar variantes cuya calidad sea menor que:

Filtrar por profundidad mínima

 Eliminar variantes cuya profundidad de cobertura sea menor que:

Filtrar por profundidad máxima

 Eliminar variantes cuya profundidad de cobertura sea mayor que:

Eliminar los indels **Mantener únicamente los indels**

Si desea añadir alguna opción adicional al comando, escribala aquí:

Primer comando que se ejecutará para hacer el filtrado de variantes:



```
vcftools --vcf ./Llamada_variantes/variantes_copy.vcf --recode --recodeINFO-all --out ./Filtrado_variantes/variantes.vcf --minQ 50 --minDP 5
```

Segundo comando que se ejecutará para obtener un informe del filtrado de variantes:



```
rtg vcfstats ./Filtrado_variantes/variantes.vcf > ./Filtrado_variantes/informe_variantes_filtradas.txt
```

Informe de resultados del filtrado de variantes:



```
Location : ./Filtrado_variantes/variantes.vcf
Failed Filters : 0
Passed Filters : 2064756
SNPs : 307710
```

10. Resumen

GenAgans

Índice

- Control de calidad de las lecturas
- Limpieza de las lecturas
- Indexado del genoma de referencia
- Mapeo
- Análisis del mapio
- Limpieza de duplicados
- Llamada de variantes
- Filtrado de variantes
- Efecto de las variantes
- Resumen**

En este apartado puede descargar todos los archivos generados hasta el momento, visualizar un resumen todos los resultados obtenidos, un resumen de todos los comandos ejecutados y descargar un script con él para lanzarlo desde la terminal si desea replicar exactamente el mismo flujo de trabajo de forma automática.

Tenga en cuenta que tanto los resultados como los comandos mostrados e incluidos en el script son los últimos que ha generado/ejecutado. Las opciones que tenga marcadas actualmente en los apartados no se reportarán aquí. De esta forma, se asegura la reproducibilidad del flujo de trabajo realizado.

Puede acceder a cada uno de estos subapartados a partir de los siguientes despliegables.

Resumen de los comandos ejecutados:

Script con los comandos ejecutados:

```
#!/bin/bash
mkdir 1.Analisis_calidad
```

[Descargar script \(.sh\)](#) [Descargar script \(.txt\)](#)

Resumen de resultados:

[Descargar todos los archivos generados](#)

Flujo de trabajo

Iconos de flujo de trabajo:

- Apartados sin informe (naranja cuadrado)
- Apartados con informe (azul cuadrado)
- Archivos (naranja cuadro)
- Flujo de trabajo (flecha negra)
- Flujo de archivos (flecha roja)

Flujo de trabajo:

```

graph LR
    A[VCF] --> B[VCF filtrado]
    B --> C[Tabla de resultados VEP]
    
```

Resaltados:

8. Filtrado de variantes (azul)

9. Efecto de las variantes (azul)

9. Efecto de las variantes

GenAgans

INICIO

1. Control de calidad de las lecturas
2. Limpieza de las lecturas
3. Individuo del genoma de referencia
4. Mapeo
5. Análisis del mapeo
6. Limpieza de duplicados
7. Llamada de variantes
8. Filtrado de variantes
9. Efecto de las variantes

Resumen

Selección sus preferencias para la predicción del efecto de las variantes:

<input checked="" type="checkbox"/> Ensamblado del genoma:	<input checked="" type="checkbox"/> Obtener significancia clínica:
GRCh38	En función del efecto
<input checked="" type="checkbox"/> Calcular SIFT	<input checked="" type="checkbox"/> Calcular PolyPhen
Clasificación + puntuación	Clasificación + Puntuación
<input checked="" type="checkbox"/> Obtener fenotipos asociados al gen afectado	<input checked="" type="checkbox"/> Obtener la clase de variante (Sequence Ontology)
<input checked="" type="checkbox"/> Obtener símbolo del gen afectado	<input checked="" type="checkbox"/> Obtener el identificador Ensembl de la proteína
<input checked="" type="checkbox"/> Obtener nomenclatura HGVS de la variante	<input checked="" type="checkbox"/> Obtener nomenclatura SPDI de la variante
<input checked="" type="checkbox"/> Obtener datos de variantes conocidas colocalizadas	
<input checked="" type="checkbox"/> Obtener referencias UniProt de proteínas colocalizadas	
<input checked="" type="checkbox"/> Obtener frecuencia alélica global de variantes colocalizadas	
<input type="checkbox"/> Limitar y ordenar los campos a incluir en la tabla:	
Si deseas añadir alguna opción adicional al comando, escribeala aquí:	

Comando que se ejecutará para predecir el efecto de las variantes:
 vep -i ./Filtrado_variantes/variantes.vcf -o
 ./Efecto_variantes/efecto_variantes.tsv --database GRCh38 --species
 homo_sapiens --force_overwrite --tab_assembly GRCh38 --clin_sig_allele 1 --sift b --polyphen b --gene_phenotype --variant_class --symbol --protein --hgvs --spdi --check_existing --uniprot --af

Tabla de resultados VEP expandida

GenAgans							
#Uploaded_variation	Location	Allele	Gene	Feature	Feature_type	Consequence	cDNA_position
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000031205	Transcript	downstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000041296	Transcript	splice_region_variant	1380
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000010008	Transcript	splice_region_variant	1120
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000010045	Transcript	splice_region_variant	1025
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000010048	Transcript	upstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000010093	Transcript	splice_region_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000010047	Transcript	splice_region_variant	358
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000010055	Transcript	downstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000011130	Transcript	upstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000011130	Transcript	upstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000011130	Transcript	upstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000011130	Transcript	splice_region_variant	408
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000011131	Transcript	upstream_gene_variant	-
L_18626498_C G	3 18626498	G	DIS00000002	ENST000000011131	Transcript	upstream_gene_variant	-



Figura 33. Esquema representativo de la interfaz y el flujo de trabajo de GenAgans desde el apartado “8. Filtrado de variantes” hasta el apartado “Resumen”. Fuente: Elaboración propia.

GenAgans: una aplicación Shiny interactiva e intuitiva para analizar variantes genéticas humanas

Por otro lado, en todas las pestañas numeradas, se añadió un botón de “Siguiente” que permanece desactivado hasta que no se han lanzado todos los comandos del apartado. Además, los botones para lanzar los comandos de los siguientes apartados también se mantienen desactivados. La activación de estos botones se programó para que fuese secuencial. De esta forma, solo se activan cuando se ha completado la ejecución del comando anterior. Esto se programó así para guiar al usuario en el flujo de trabajo y asegurar su correcto funcionamiento ([Anexo 15](#)).

Finalmente, la aplicación se probó con dos conjuntos de datos experimentales públicos. Por una parte, con el primer conjunto de datos se lanzaron todos los comandos con las opciones disponibles a excepción del filtrado de indels y SNVs, el filtrado por profundidad máxima y la limitación de los campos de la tabla de VEP ([Anexo 14](#)). Esto se hizo para probar las opciones y, al mismo tiempo, no limitar los resultados finales, como si de buscar posibles biomarcadores se tratase. Los comandos se pudieron lanzar de forma secuencial e intuitiva, comprobando el comando exacto a ejecutar en cada caso. En cada apartado se pudieron visualizar con comodidad los resultados obtenidos. Tras la llamada de variantes y su filtrado se identificaron 870 variantes que, posteriormente, fueron anotadas con VEP. No se detectó ningún error en el programa, por lo que en último lugar se obtuvo una tabla con las anotaciones para cada una de las variantes identificadas y se descargó el script personalizado ([Fig. A14.20-24](#)). El script y los informes de resultados generados en esta ejecución se adjuntan como ejemplo en el [repositorio de GitHub](#) para una visualización más cómoda.

Adicionalmente, con el segundo conjunto de datos se comprobó la versatilidad de GenAgans, reproduciendo el flujo de trabajo descrito en el apartado de métodos del artículo del que provienen las muestras (53). El programa permitió su reproducción, incluyendo la personalización de los comandos con las opciones indicadas en el artículo ([Fig. 31-33](#)).

A lo largo del desarrollo de la aplicación se ha buscado proporcionar una herramienta que simplifique y facilite el flujo de trabajo de identificación y predicción del efecto de las variantes genéticas en muestras de humanos. En comparación con otras aplicaciones más complejas, como Galaxy (49), por su gran variedad de herramientas independientes, GenAgans ofrece una serie de herramientas concretas que se disponen de forma secuencial. De forma que los archivos de salida de una herramienta se concatenan

automáticamente con los de entrada de la siguiente. Así, en GenAgans se ha eliminado la problemática de la gestión de archivos y la toma de decisiones sobre qué herramientas emplear en el flujo de trabajo.

Del mismo modo, en comparación con herramientas como Taverna (51), que requiere conocimientos avanzados para configurar flujos de trabajo personalizados, la UI interactiva de GenAgans permite a los usuarios realizar los análisis sin la necesidad de dichos conocimientos. Esto se consigue gracias a la disposición de un flujo de trabajo preestablecido y comprobado.

Finalmente, a diferencia de herramientas como Snakemake (50), donde es imprescindible tener conocimientos en programación y que no presenta UI, GenAgans ofrece una UI interactiva e intuitiva. De forma que cada etapa del flujo está claramente guiada y explicada para que el usuario conozca en cada momento qué comando se va a ejecutar, por qué y qué esperar como resultado. Esto facilita el acceso al flujo de trabajo de análisis de variantes genéticas humanas a un público mucho más amplio.

Todas estas mejoras en accesibilidad se integran en un conjunto de herramientas y un flujo de trabajo robusto que, aunque no ofrece la misma versatilidad que otros programas, permite la personalización de cada etapa del proceso y garantiza tanto su reproducibilidad como su posterior automatización.

5. Conclusiones

Tras el desarrollo y comprobación del funcionamiento de GenAgans, se pueden extraer las siguientes conclusiones basadas en los objetivos del trabajo:

1. Mediante el uso de Shiny para Python y la implementación de diversas herramientas bioinformáticas, se ha logrado desarrollar GenAgans. Una aplicación que facilita el flujo de trabajo para la identificación y predicción del efecto de variantes genéticas en humanos.
2. Se consiguió añadir diversas opciones que permiten personalizar el flujo de trabajo en función de las necesidades del usuario.

3. Los archivos de salida de los comandos se enlazaron directamente como archivos de entrada del siguiente comando durante las diversas etapas del flujo de trabajo. Por tanto, se consiguió la automatización del flujo de trabajo.
4. GenAgans genera un informe final con las variantes identificadas y sus anotaciones.
5. El diseño intuitivo de la UI permite guiar al usuario durante todo el flujo de trabajo de forma interactiva, intuitiva y controlada.
6. GenAgans permite la visualización en la propia UI de los diferentes informes de resultados que se generan a lo largo del flujo de trabajo.
7. Para cada uno de los pasos del flujo de trabajo se programó un botón para la descarga de los archivos generados. Además, se incluyó la opción de descargar todos los archivos generados durante el flujo de trabajo en el apartado final.
8. Se programó la generación y descarga de un script personalizado con todos los comandos ejecutados por el usuario, incluyendo las opciones seleccionadas en el momento de su ejecución.
9. Junto al código disponible en GitHub se proporciona también un archivo YAML para la rápida instalación del ambiente de Conda que contiene todas las herramientas y dependencias necesarias para ejecutar GenAgans.

6. Limitaciones y perspectivas futuras

Si bien GenAgans ha cumplido con todos los objetivos propuestos, existen algunas limitaciones que se abordarán en un futuro.

La principal limitación de la aplicación es el gran coste computacional y de almacenamiento que supone su uso para el procesamiento de grandes cantidades de datos como en estudios de tipo WGS. Esto supone una limitación porque requiere de un ordenador con una gran capacidad de memoria para poder realizar el flujo de trabajo en segundo plano sin consumir toda la memoria RAM o paralizar otras funciones. Como solución a este problema, se

plantea subir la aplicación a un servidor web con un dominio propio que maneje la carga computacional, en lugar de tener que ejecutarse en local.

Por último, otra limitación es la gran cantidad de tiempo que supone el indexado del genoma de referencia. Este puede tardar desde varios minutos hasta varias horas en función del tamaño del archivo. Teniendo en cuenta que estos archivos de indexado solo requieren generarse una vez y luego pueden reutilizarse, se plantea crear en un futuro diversas versiones de GenAgans que adjunten directamente los archivos de indexado de diferentes cromosomas y versiones del genoma de referencia humano para evitar dicho proceso.

7. Referencias bibliográficas

1. Mardis ER. A decade's perspective on DNA sequencing technology. *Nature* [Internet]. 2011;470(7333):198–203. Available from: <http://dx.doi.org/10.1038/nature09796>
2. Brittain HK, Scott R, Thomas E. The rise of the genome and personalised medicine. *Clin Med J R Coll Physicians London*. 2017;17(6):545–51.
3. Roy S, Coldren C, Karunamurthy A, Kip NS, Klee EW, Lincoln SE, et al. Standards and Guidelines for Validating Next-Generation Sequencing Bioinformatics Pipelines: A Joint Recommendation of the Association for Molecular Pathology and the College of American Pathologists. *J Mol Diagnostics*. 2018;20(1):4–27.
4. Dayhoff MO, Ledley RS. Comprotein: a computer program to aid primary protein structure determination. In: Proceedings of the December 4-6, 1962, Fall Joint Computer Conference. New York, USA: ACM Press; 1962. p. 262–74.
5. Gauthier J, Vincent AT, Charette SJ, Derome N. A brief history of bioinformatics. *Brief Bioinform*. 2019;20(6):1981–96.
6. Sanger F, Nicklen S, Coulson AR. DNA sequencing with chain-terminating inhibitors. *Proc Natl Acad Sci U S A*. 1977;74(12):5463–7.
7. Slatko BE, Gardner AF, Ausubel FM. Overview of Next-Generation Sequencing Technologies. *Curr Protoc Mol Biol*. 2018 Apr 16;122(1):1–11.
8. Maxam AM, Gilbert W. A new method for sequencing DNA. *Proc Natl Acad Sci U S A*. 1977;74(2):560–4.
9. Eren K, Taktakoglu N, Pirim I. DNA Sequencing Methods: From Past to Present. *Eurasian J Med*. 2023 Jan 18;54(Supp1):S47–56.

10. Sanger F, Air GM, Barrell BG, Brown NL, Coulson AR, Fiddes CA, et al. Nucleotide sequence of bacteriophage phi X174 DNA. *Nature*. 1977 Feb 24;265(5596):687–95.
11. Zhong Y, Xu F, Wu J, Schubert J, Li MM. Application of Next Generation Sequencing in Laboratory Medicine. *Ann Lab Med*. 2021 Jan;41(1):25–43.
12. Satam H, Joshi K, Mangrolia U, Waghoo S, Zaidi G, Rawool S, et al. Next-Generation Sequencing Technology: Current Trends and Advancements. *Biology (Basel)*. 2023;12(7).
13. Athanasopoulou K, Boti MA, Adamopoulos PG, Skourou PC, Scorilas A. Third-Generation Sequencing: The Spearhead towards the Radical Transformation of Modern Genomics. *Life*. 2021 Dec 26;12(1).
14. Schadt EE, Linderman MD, Sorenson J, Lee L, Nolan GP. Computational solutions to large-scale data management and analysis. *Nat Rev Genet*. 2010;11(9):647–57.
15. Vailati-Riboni M, Palombo V, Loor JJ. What Are Omics Sciences? In: Periparturient Diseases of Dairy Cows. Cham: Springer International Publishing; 2017. p. 1–7.
16. Martínez-Espinosa R. Impact of the “Omics Sciences” in Medicine: New Era for Integrative Medicine. *J Clin Microbiol Biochem Technol*. 2017;3:009–13.
17. World Health Organization. Genomics [Internet]. 2024 [cited 2024 Aug 2]. Available from: https://www.who.int/health-topics/genomics#tab=tab_1
18. Bohlander SK. ABCs of genomics. *Hematology*. 2013 Dec 6;2013(1):316–23.
19. Gonzaga-Jauregui C, Lupski JR, Gibbs RA. Human Genome Sequencing in Health and Disease. *Annu Rev Med*. 2012 Feb 18;63(1):35–61.
20. Berger MF, Mardis ER. The emerging clinical relevance of genomics in cancer medicine. *Nat Rev Clin Oncol*. 2018 Jun 29;15(6):353–65.
21. Marian AJ. Clinical Interpretation and Management of Genetic Variants. *JACC Basic to Transl Sci*. 2020;5(10):1029–42.
22. Dhindsa RS, Wang Q, Vitsios D, Burren OS, Hu F, DiCarlo JE, et al. A minimal role for synonymous variation in human disease. *Am J Hum Genet*. 2022 Dec;109(12):2105–9.
23. Sun H, Yu G. New insights into the pathogenicity of non-synonymous variants through multi-level analysis. *Sci Rep*. 2019;9(1):1–11.
24. Gerasimavicius L, Livesey BJ, Marsh JA. Loss-of-function, gain-of-function and dominant-negative mutations have profoundly different effects on protein structure. *Nat Commun*. 2022;13(1):1–15.
25. Pei XM, Yeung MHY, Wong ANN, Tsang HF, Yu ACS, Yim AKY, et al. Targeted

- Sequencing Approach and Its Clinical Applications for the Molecular Diagnosis of Human Diseases. *Cells*. 2023;12(3).
- 26. Brlek P, Bulić L, Bračić M, Projić P, Škaro V, Shah N, et al. Implementing Whole Genome Sequencing (WGS) in Clinical Practice: Advantages, Challenges, and Future Perspectives. *Cells*. 2024 Mar 13;13(6):504.
 - 27. Richards S, Aziz N, Bale S, Bick D, Das S, Gastier-Foster J, et al. Standards and guidelines for the interpretation of sequence variants: A joint consensus recommendation of the American College of Medical Genetics and Genomics and the Association for Molecular Pathology. *Genet Med*. 2015;17(5):405–24.
 - 28. Li MM, Datto M, Duncavage EJ, Kulkarni S, Lindeman NI, Roy S, et al. Standards and Guidelines for the Interpretation and Reporting of Sequence Variants in Cancer: A Joint Consensus Recommendation of the Association for Molecular Pathology, American Society of Clinical Oncology, and College of American Pathologists. *J Mol Diagnostics*. 2017;19(1):4–23.
 - 29. Spielmann M, Kircher M. Computational and experimental methods for classifying variants of unknown clinical significance. *Cold Spring Harb Mol Case Stud*. 2022;8(3).
 - 30. National Center for Biotechnology Information (NCBI). ClinVar [Internet]. 2024. Available from: <https://www.ncbi.nlm.nih.gov/clinvar/>
 - 31. Online Mendelian Inheritance in Man (OMIM). OMIM - Online Mendelian Inheritance in Man [Internet]. 2024. Available from: <https://www.omim.org/>
 - 32. National Center for Biotechnology Information (NCBI). dbSNP - Single Nucleotide Polymorphism Database [Internet]. 2024. Available from: <https://www.ncbi.nlm.nih.gov/snp/>
 - 33. Ensembl. Ensembl Genome Browser [Internet]. 2024. Available from: <https://www.ensembl.org/index.html>
 - 34. Federici G, Soddu S. Variants of uncertain significance in the era of high-throughput genome sequencing: A lesson from breast and ovary cancers. *J Exp Clin Cancer Res*. 2020;39(1):1–12.
 - 35. Jaganathan K, Kyriazopoulou Panagiotopoulou S, McRae JF, Darbandi SF, Knowles D, Li YI, et al. Predicting Splicing from Primary Sequence with Deep Learning. *Cell*. 2019;176(3):535-548.e24.
 - 36. Sim N-L, Kumar P, Hu J, Henikoff S, Schneider G, Ng PC. SIFT web server: predicting effects of amino acid substitutions on proteins. *Nucleic Acids Res*. 2012

- Jul 1;40(W1):W452–7.
37. Ramensky V, Bork P, Sunyaev S. Human non-synonymous SNPs: server and survey. *Nucleic Acids Res.* 2002 Sep 1;30(17):3894–900.
 38. Flanagan SE, Patch A-M, Ellard S. Using SIFT and PolyPhen to Predict Loss-of-Function and Gain-of-Function Mutations. *Genet Test Mol Biomarkers.* 2010 Aug;14(4):533–7.
 39. Katsonis P, Wilhelm K, Williams A, Lichtarge O. Genome interpretation using in silico predictors of variant impact. *Hum Genet.* 2022;141(10):1549–77.
 40. Iancu IF, Avila-Fernandez A, Arteche A, Trujillo-Tiebas MJ, Riveiro-Alvarez R, Almoguera B, et al. Prioritizing variants of uncertain significance for reclassification using a rule-based algorithm in inherited retinal dystrophies. *npj Genomic Med.* 2021;6(1).
 41. Wain KE, Azzariti DR, Goldstein JL, Johnson AK, Krautscheid P, Lepore B, et al. Variant interpretation is a component of clinical practice among genetic counselors in multiple specialties. *Genet Med.* 2020;22(4):785–92.
 42. Wang YC, Wu Y, Choi J, Allington G, Zhao S, Khanfar M, et al. Computational Genomics in the Era of Precision Medicine: Applications to Variant Analysis and Gene Therapy. *J Pers Med.* 2022;12(2).
 43. Dehghan A. Genome-wide association studies. *Methods Mol Biol.* 2018;1793:37–49.
 44. Kudela E, Samec M, Kubatka P, Nachajova M, Laucekova Z, Liskova A, et al. Breast cancer in young women: Status quo and advanced disease management by a predictive, preventive, and personalized approach. *Cancers (Basel).* 2019;11(11):1–20.
 45. Kolesar J, Peh S, Thomas L, Baburaj G, Mukherjee N, Kantamneni R, et al. Integration of liquid biopsy and pharmacogenomics for precision therapy of EGFR mutant and resistant lung cancers. *Mol Cancer.* 2022;21(1):1–22.
 46. SoRelle JA, Wachsmann M, Cantarel BL. Assembling and Validating Bioinformatic Pipelines for Next-Generation Sequencing Clinical Assays. *Arch Pathol Lab Med.* 2020 Sep 1;144(9):1118–30.
 47. Sohail A, Arif F. Supervised and unsupervised algorithms for bioinformatics and data science. *Prog Biophys Mol Biol.* 2020;151:14–22.
 48. Kouskoumvekaki I, Shublaq N, Brunak S. Facilitating the use of large-scale biological data and tools in the era of translational bioinformatics. *Brief Bioinform.* 2013;15(6):942–52.

49. Afgan E, Baker D, Batut B, van den Beek M, Bouvier D, Čech M, et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Res.* 2018 Jul 2;46(W1):W537–44.
50. Köster J, Rahmann S. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics* [Internet]. 2012 Oct 1;28(19):2520–2. Available from: <https://academic.oup.com/bioinformatics/article/28/19/2520/290322>
51. Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*. 2004 Nov 22;20(17):3045–54.
52. Attwood TK, Blackford S, Brazas MD, Davies A, Schneider MV. A global perspective on evolving bioinformatics and data science training needs. *Brief Bioinform.* 2019;20(2):398–404.
53. Singh V, Katiyar A, Malik P, Kumar S, Mohan A, Singh H, et al. Identification of molecular biomarkers associated with non-small-cell lung carcinoma (NSCLC) using whole-exome sequencing. *Cancer Biomarkers.* 2023;1:1–18.
54. European Nucleotide Archive. Project: PRJNA734015 [Internet]. 2024. Available from: <https://www.ebi.ac.uk/ena/browser/view/PRJNA734015>
55. Ensembl. Ensembl Genome Browser GRCh37 [Internet]. 2024. Available from: https://ftp.ensembl.org/pub/grch37/current/fasta/homo_sapiens/dna/
56. Ensembl. Ensembl Genome Browser Release 112 [Internet]. 2024. Available from: https://ftp.ensembl.org/pub/release-112/fasta/homo_sapiens/dna/
57. Andrews S. FastQC: A Quality Control Tool for High Throughput Sequence Data [Internet]. Babraham Bioinformatics; 2010. Available from: <https://www.bioinformatics.babraham.ac.uk/projects/fastqc/>
58. Chen S, Zhou Y, Chen Y, Gu J. Fastp: An ultra-fast all-in-one FASTQ preprocessor. *Bioinformatics.* 2018;34(17):i884–90.
59. Li H, Durbin R. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics.* 2009;25(14):1754–60.
60. Li H, Durbin R. Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics.* 2010;26(5):589–95.
61. Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, et al. The Sequence Alignment/Map format and SAMtools. *Bioinformatics.* 2009;25(16):2078–9.
62. Broad Institute. Picard Toolkit [Internet]. 2019. Available from: <https://broadinstitute.github.io/picard/>

63. Garrison E, Marth G. Haplotype-based variant detection from short-read sequencing. 2012;1–9. Available from: <http://arxiv.org/abs/1207.3907>
64. Cleary JG, Braithwaite R, Gaastra K, Hilbush BS, Inglis S, Irvine SA, et al. Joint variant and de novo mutation identification on pedigrees from high-throughput sequencing data. *J Comput Biol.* 2014;21(6):405–19.
65. Danecek P, Auton A, Abecasis G, Albers CA, Banks E, DePristo MA, et al. The variant call format and VCFtools. *Bioinformatics.* 2011;27(15):2156–8.
66. McLaren W, Gil L, Hunt SE, Riat HS, Ritchie GRS, Thormann A, et al. The Ensembl Variant Effect Predictor. *Genome Biol.* 2016;17(1):1–14.
67. Koboldt DC. Best practices for variant calling in clinical sequencing. *Genome Med.* 2020;12(1):1–13.
68. He X, Chen S, Li R, Han X, He Z, Yuan D, et al. Comprehensive fundamental somatic variant calling and quality management strategies for human cancer genomes. *Brief Bioinform.* 2021;22(3):1–15.
69. Posit. Shiny for Python [Internet]. Available from: <https://github.com/posit-dev/py-shiny>
70. Posit. Shiny Core API [Internet]. Available from: <https://shiny.posit.co/py/api/core/>

8. Anexos

Anexo 1. Script para el análisis de variantes genéticas en humano

```
#!/bin/bash
```

```
mkdir 1.Analisis_calidad
```

```
fastqc -o 1.Analisis_calidad/ SRR14695036_1.fastq.gz SRR14695036_2.fastq.gz
```

```
mkdir 2.Lecturas_limpias
```

```
fastp -i SRR14695036_1.fastq.gz -I SRR14695036_2.fastq.gz -o  
.2.Lecturas_limpias/SRR14695036_1_clean.fq.gz -O  
.2.Lecturas_limpias/SRR14695036_2_clean.fq.gz --cut_tail 15 --cut_front 15 --  
cut_mean_quality 30 --detect_adapter_for_pe --trim_poly_g --trim_poly_x -l 100 -h  
.2.Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.html -j  
.2.Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.json
```

```
mkdir 3.Genoma_referencia_indexado
```

```
cp Homo_sapiens.GRCh37.dna.chromosome.1.fa
```

```
.3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa
```

```
bwa index -p
```

```
.3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa
```

```
.3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa
```

```
mkdir 4.Mapeo
```

```
bwa mem -a
```

```
.3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa
```

```
./2.Lecturas_limpias/SRR14695036_1_clean.fq.gz  
./2.Lecturas_limpias/SRR14695036_2_clean.fq.gz -o ./4.Mapeo/mapeo.sam
```

```
mkdir 5.Analisis_mapeo
```

```
samtools view -bS ./4.Mapeo/mapeo.sam -o ./5.Analisis_mapeo/mapeo.bam
```

```
samtools sort ./5.Analisis_mapeo/mapeo.bam -o ./5.Analisis_mapeo/mapeo_sorted.bam
```

```
samtools stats ./5.Analisis_mapeo/mapeo_sorted.bam >  
./5.Analisis_mapeo/analisis_mapeo.txt
```

```
plot-bamstats -p ./5.Analisis_mapeo/informe_mapeo  
./5.Analisis_mapeo/analisis_mapeo.txt
```

```
mkdir 6.Limpieza_duplicados
```

```
picard MarkDuplicates --INPUT ./5.Analisis_mapeo/mapeo_sorted.bam --OUTPUT  
.6.Limpieza_duplicados/mapeo_dedup.bam --METRICS_FILE  
.6.Limpieza_duplicados/MarkDuplicatesMetrics.txt --ASSUME_SORTED True
```

```
picard AddOrReplaceReadGroups -I ./6.Limpieza_duplicados/mapeo_dedup.bam  
-O ./6.Limpieza_duplicados/mapeo_dedup_RG.bam -RGID default -RGLB lib1 -RGPL  
ILLUMINA -RGSM sample1 -RGPU unit1
```

```
mkdir 7.Llamada_variantes
```

```
mv Homo_sapiens.GRCh37.dna.chromosome.19.fa  
.7.Llamada_variantes/Homo_sapiens.GRCh37.dna.chromosome.19.fa
```

```
samtools faidx ./7.Llamada_variantes/Homo_sapiens.GRCh37.dna.chromosome.19.fa
```

```
samtools index ./6.Limpieza_duplicados/mapeo_dedup_RG.bam
```

```
freebayes --min-alternate-count 50 -f  
.7.Llamada_variantes/Homo_sapiens.GRCh37.dna.chromosome.19.fa  
.6.Limpieza_duplicados/mapeo_dedup_RG.bam > ./7.Llamada_variantes/variantes.vcf
```

```
rtg vcfstats ./7.Llamada_variantes/variantes.vcf >  
.7.Llamada_variantes/informe_variantes.txt
```

```
mkdir 8.Filtrado_variantes
```

```
vcftools --vcf ./7.Llamada_variantes/variantes.vcf --recode --recode-INFO-all --out  
.8.Filtrado_variantes/variantes_filtradas.vcf --remove-indels
```

```
mv ./8.Filtrado_variantes/variantes_filtradas.vcf.recode.vcf  
.8.Filtrado_variantes/variantes_filtradas.vcf
```

```
rtg vcfstats ./8.Filtrado_variantes/variantes_filtradas.vcf >  
.8.Filtrado_variantes/informe_variantes_filtradas.txt
```

```
mkdir 9.Efecto_variantes
```

```
vep -i ./8.Filtrado_variantes/variantes_filtradas.vcf -o  
.9.Efecto_variantes/efecto_variantes.tsv --database --species homo_sapiens --  
force_overwrite --tab --assembly GRCh37
```

Anexo 2. Sistema de carpetas creado con el script

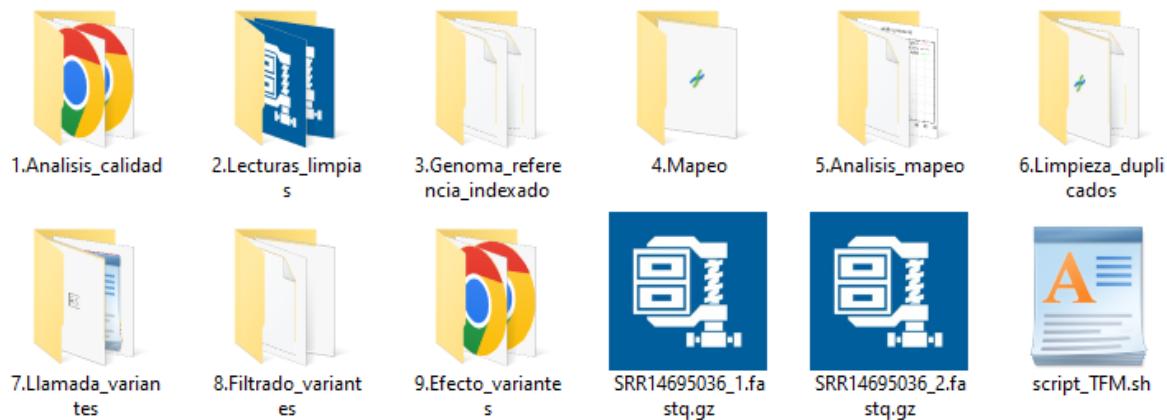


Figura A2. Sistema de carpetas creado automáticamente tras ejecutar el script. Fuente: Elaboración propia.

Anexo 3. Código para la definición de la UI del apartado “Inicio”

Líneas de código: 15 a 67.

```
Inicio = ui.page_fluid(  
    ui.layout_column_wrap(  
        ui.value_box(  
            "¡Bienvenido/a!",  
            ui.output_ui("texto_introduccion"),  
            showcase=icon_svg("dna"),  
        ),  
    ),  
    ui.layout_column_wrap(  
        ui.value_box(  
            "1r archivo de lecturas",  
            ui.output_text("archivo1_nombre"),  
            height="150px",  
            showcase=icon_svg("file"),  
        ),  
        ui.value_box(  
            "2o archivo de lecturas",  
            ui.output_text("archivo2_nombre"),  
            height="150px",  
            showcase=icon_svg("file"),  
        ),  
        ui.value_box(  
            "Genoma de referencia",  
            ui.output_text("archivo3_nombre"),  
            height="150px",  
            showcase=icon_svg("book"),  
        ),  
    ),  
    ui.layout_column_wrap(  
        ui.value_box(  
            "Cargue el primer archivo de lecturas",  
            ui.input_file("archivo1", "", multiple=False, accept=[ ".fastq.gz", ".fq.gz"],  
                         button_label="Cargar...", placeholder="Vacío"),  
            height="150px",  
            showcase=icon_svg("upload"),  
        ),  
        ui.value_box(  
            "Cargue el segundo archivo de lecturas",  
            ui.input_file("archivo2", "", multiple=False, accept=[ ".fastq.gz", ".fq.gz"],  
                         button_label="Cargar...", placeholder="Vacío"),  
            height="150px",  
            showcase=icon_svg("upload"),  
        ),  
    ),  
)
```

```
        button_label="Cargar...", placeholder="Vacio"),
        showcase=icon_svg("upload"),
    ),
    ui.value_box(
        "Cargue el genoma de referencia para el mapeo",
        ui.input_file("archivo3", "", multiple=False, accept=[".fasta", ".fa"],
                      button_label="Cargar...", placeholder="Vacio"),
        showcase=icon_svg("upload"),
    ),
),
ui.layout_column_wrap(
    ui.input_action_button("boton_empezar", "EMPEZAR", disabled=True),
),
)
```

Anexo 4. Comparativa de uso entre ui.layout_columns() y ui.layout_column_wrap()

a

Si desea añadir Read Groups marque esta casilla, puede usar los valores predefinidos o introducir los propios

RGID - ID del grupo de lecturas (campo ID): default	RGLB - Nombre de la librería (campo LB): lib1	RGPL - Nombre del secuenciador (campo PL): NA	RGSM - Nombre de la muestra (campo SM): sample1	RGPU - Nombre de la carrera (run) (campo PU): unit1
--	--	--	--	--

> Comando que se ejecutará para añadir los Read Groups:
Cargue los archivos, por favor.

Iniciar adición de Read Groups

Descargar resultados Siguiente

b

Si desea añadir Read Groups marque esta casilla, puede usar los valores predefinidos o introducir los propios

RGID - ID del grupo de lecturas (campo ID): default	RGLB - Nombre de la librería (campo LB): lib1	RGPL - Nombre del secuenciador (campo PL): NA	RGSM - Nombre de la muestra (campo SM): sample1
RGPU - Nombre de la carrera (run) (campo PU): unit1			

> Comando que se ejecutará para añadir los Read Groups:
Cargue los archivos, por favor.

Iniciar adición de Read Groups

Descargar resultados Siguiente

Figura A4. Comparativa entre la disposición de los componentes del panel de opciones de RG en función del uso de ui.layout_columns() (a) o ui.layout_column_wrap() (b). Fuente: elaboración propia.

Anexo 5. Ejemplo de uso de ui.input_checkbox(), ui.input_numeric() y ui.input_text() en el código UI para el apartado “2. Limpieza de las lecturas”

```
114 ui.layout_columns(
115   ui.card("Seleccione sus preferencias para la limpieza de las lecturas:",
116     ui.layout_columns(
117       ui.card(ui.input_checkbox("check_Q_media", "Restringir calidad media de las lecturas", True),
118         ui.input_numeric("Q_media", "Eliminar lecturas que tengan una calidad media menor que:", 30, min=1, width="100%")),
119       ui.card(ui.input_checkbox("check_Q_inicio", "Recortar bases en el extremo 5' (inicio) de las lecturas", True),
120         ui.input_numeric("Q_inicio", "Número de bases a eliminar al inicio:",
121           15, min=1, width="100%")),
122       ui.card(ui.input_checkbox("check_Q_final", "Recortar bases en el extremo 3' (final) de las lecturas", True),
123         ui.input_numeric("Q_final", "Número de bases a eliminar al final:",
124           15, min=1, width="100%")),
125       ui.card(ui.input_checkbox("check_longitud", "Restringir la longitud mínima de las lecturas", True),
126         ui.input_numeric("longitud", "Eliminar lecturas cuya longitud sea menor que:", 100, min=1, width="100%")),
127     ),
128     ui.layout_columns(
129       ui.card(ui.input_checkbox("check_adaptadores", "Eliminar adaptadores", True),),
130       ui.card(ui.input_checkbox("check_polyG", "Eliminar secuencias polyG", True),),
131       ui.card(ui.input_checkbox("check_polyX", "Eliminar secuencias polyX", True),),
132     ),
133     ui.layout_columns(
134       ui.card(ui.input_text("opcion_adicional2", "Si desea añadir alguna opción adicional al comando, escribala aquí:",
135         True, width="100%")),
136     ),
137   ),
138 )
```

Figura A5. Código UI para definir el panel de opciones del apartado 2. Fuente: Elaboración propia.

Anexo 6. Apariencia del panel de opciones del apartado “2. Limpieza de las lecturas”

Seleccione sus preferencias para la limpieza de las lecturas:

Restringir calidad media de las lecturas

Eliminar lecturas que tengan una calidad media menor que:

30

Recortar bases en el extremo 5' (inicio) de las lecturas

Número de bases a eliminar al inicio:

15

Recortar bases en el extremo 3' (final) de las lecturas

Número de bases a eliminar al final:

15

Restringir la longitud mínima de las lecturas

Eliminar lecturas cuya longitud sea menor que:

100

Eliminar adaptadores

Eliminar secuencias polyG

Eliminar secuencias polyX

Si desea añadir alguna opción adicional al comando, escríbala aquí:

Figura A6. UI para el panel de opciones del apartado 2. Fuente: Elaboración propia.

Anexo 7. Código UI para las opciones del apartado “9. Efecto de las variantes”

```

467 Efecto_variantes = ui.page_fluid(
468     ui.layout_columns(
469         ui.card("Seleccione sus preferencias para la predicción del efecto de las variantes:",
470             ui.layout_columns(
471                 ui.card(ui.tooltip(ui.input_checkbox("check_assembly"), "Ensamblado del genoma:", True),
472                     "Si no se indica, VEP tratará de inferirlo. Si no lo consigue, asumirá el GRCh38.",  

473                     id="tooltip_assembly", placement="right"),
474                     ui.input_select("seleccion_assembly", "", {"GRCh37": "GRCh37", "GRCh38": "GRCh38"},  

475                     selected="GRCh37", width="100%")),
476                 ui.card(ui.tooltip(ui.input_checkbox("check_clin_sig_allele"), "Obtener significancia clínica:", False),
477                     'En función de la posición genómica: Devuelve toda la información de \  

478                     significancia clínica conocida para la posición genómica de la variante, independientemente del alelo. \  

479                     En función del alelo: Devuelve solo la información de significancia \  

480                     clínica conocida para el alelo en el que se encuentra la variante',  

481                     id="tooltip_clin_sig", placement="right"),
482                     ui.input_select("seleccion_clin_sig", "",  

483                         {"0": "En función de la posición genómica", "1": "En función del alelo"},  

484                         selected="1", width="100%")),
485             ),
486             ui.layout_columns(
487                 ui.card(ui.tooltip(ui.input_checkbox("check_sift"), "Calcular SIFT", False),
488                     'Clasificación: Asigna una categoría a la variante en función de la predicción \  

489                     (ej. "Tolerada" o "Dañina"). Puntuación: muestra la puntuación entre 0 y 1, siendo cercano a 0 \  

490                     "probablemente dañina" y cercano a 1 "probablemente tolerada".',
491                     id="tooltip_sift", placement="right"),
492                     ui.input_select("seleccion_sift", "",  

493                         {"p": "Clasificación", "s": "Puntuación", "b": "Clasificación + puntuación"},  

494                         selected="b", width="100%")),
495                 ui.card(ui.tooltip(ui.input_checkbox("check_polyphen"), "Calcular PolyPhen", False),
496                     'Clasificación: Asigna una categoría a la variante en función de la predicción \  

497                     (ej. "Benigna" o "Dañina"). Puntuación: muestra la puntuación entre 0 y 1, siendo cercano a 0 \  

498                     "probablemente benigna" y cercano a 1 "probablemente dañina".',
499                     id="tooltip_polyphen", placement="right"),
500                     ui.input_select("seleccion_polyphen", "",  

501                         {"p": "Clasificación", "s": "Puntuación", "b": "Clasificación + Puntuación"},  

502                         selected="b", width="100%")),
503             ),
504             ui.layout_columns(
505                 ui.input_checkbox("check_gene_phenotype", "Obtener fenotipos asociados al gen afectado",
506                     False, width="100%")),
507                 ui.input_checkbox("check_variant_class", "Obtener la clase de variante (Sequence Ontology)", False, width="100%")),
508             ),
509             ui.layout_columns(
510                 ui.input_checkbox("check_symbol", "Obtener símbolo del gen afectado", False, width="100%")),
511                 ui.input_checkbox("check_protein", "Obtener el identificador Ensembl de la proteína", False, width="100%")),
512             ),
513             ui.layout_columns(
514                 ui.input_checkbox("check_hgvs", "Obtener nomenclatura HGVS de la variante", False, width="100%")),
515                 ui.input_checkbox("check_spdi", "Obtener nomenclatura SPDI de la variante", False, width="100%")),
516             ),
517             ui.layout_columns(
518                 ui.input_checkbox("check_check_existing", "Obtener datos de variantes conocidas colocalizadas", False, width="100%")),
519             ),
520             ui.layout_columns(
521                 ui.input_checkbox("check_uniprot", "Obtener referencias UniProt de proteínas colocalizadas", False, width="100%")),
522             )

```



Figura A7.1. Código UI para el panel de opciones del apartado 9. #1. Fuente: Elaboración propia.



```
523 |         ui.layout_columns(
524 |             ui.card(ui.input_checkbox("check_af", "Obtener frecuencia alélica global de variantes colocalizadas", False, width="100%")),
525 |         ),
526 |         ui.layout_columns(
527 |             ui.input_checkbox("check_fields", "Limitar y ordenar los campos a incluir en la tabla:", False, width="100%"),
528 |             ui.input_selectize("campos_seleccionados", "",
529 |                 ("Uploaded_variation", "Location", "Allele", "Gene", "Feature", "Feature_type", "Consequence",
530 |                  "cDNA_position", "CDS_position", "Protein_position", "Amino_acids", "Codons", "Existing_variation",
531 |                  "VARIANT_CLASS", "SIFT", "PolyPhen", "NEAREST", "GENE_PHENO", "MOTIF_NAME", "MOTIF_POS", "HIGH_INF_POS",
532 |                  "MOTIF_SCORE_CHANGE", "CELL_TYPE", "IND", "ZYG", "ALLELE_NUM", "REF_ALLELE", "UPLOADED_ALLELE",
533 |                  "EXON", "INTRON", "HGVSc", "HGVSp", "HGVs_OFFSET", "HGVsg", "SPDI", "ENSP", "SYMBOL", "SYMBOL_SOURCE",
534 |                  "HGNC_ID", "CCDS", "SWISSPROT", "TREMBL", "UNIPARC", "UNIPROT_ISOFORM", "TSL", "APPRIS", "CANONICAL",
535 |                  "MANE_SELECT", "MANE_PLUS_CLINICAL", "BIOTYPE", "DOMAINS", "RefSeq", "CLIN_SIG", "SOMATIC", "PHENO",
536 |                  "SV", "AF", "CHECK_REF", "PICK", "FREQS"),
537 |                 multiple=True, width="100%")),
538 |         ),
539 |         ui.layout_columns(
540 |             ui.card(ui.input_text("opcion_adicional9", "Si desea añadir alguna opción adicional al comando, escribeala aquí:", True, width="100%")),
541 |         ),
542 |     ),
543 | )
```

Figura A7.2. Código UI para el panel de opciones del apartado 9. #2. Fuente: Elaboración propia.

Anexo 8. Apariencia de las opciones y el comando del apartado “9. Efecto de las variantes”

GenAgans

The screenshot shows the '9. Efecto de las variantes' section of the GenAgans application. On the left, a sidebar lists various steps: Inicio, Control de calidad de las lecturas, Limpieza de las lecturas, Indexado del genoma de referencia, Mapeo, Análisis del mapeo, Limpieza de duplicados, Llamada de variantes, Filtrado de variantes, and Efecto de las variantes (which is highlighted in teal). Below these are Resumen and Salida.

The main panel is titled 'Seleccione sus preferencias para la predicción del efecto de las variantes'. It contains several sections with checkboxes and dropdown menus:

- Ensamblado del genoma:** GRCh37 (selected) / GRCh38
- Obtener significancia clínica:** En función del alelo
- Calcular SIFT:** Clasificación + puntuación
- Calcular PolyPhen:** Clasificación + Puntuación
- Obtener fenotipos asociados al gen afectado**
- Obtener la clase de variante (Sequence Ontology)**
- Obtener símbolo del gen afectado**
- Obtener el identificador Ensembl de la proteína**
- Obtener nomenclatura HGVS de la variante**
- Obtener nomenclatura SPDI de la variante**
- Obtener datos de variantes conocidas colocalizadas**
- Obtener referencias UniProt de proteínas colocalizadas**
- Obtener frecuencia alélica global de variantes colocalizadas**
- Limitar y ordenar los campos a incluir en la tabla:** Uploaded_variation x Location x

A tooltip appears over the 'Ensamblado del genoma' checkbox: 'Si no se indica, VEP tratará de inferirlo. Si no lo consigue, asumirá el GRCh38.'

Below the checkboxes is a text input field for additional command options: 'Si desea añadir alguna opción adicional al comando, escríbala aquí:' followed by a large empty text area.

At the bottom, a terminal-like icon (>) is followed by the command: 'Comando que se ejecutará para predecir el efecto de las variantes: vep -i ./8.Filtrado_variantes/variantes.vcf -o ./9.Efecto_variantes/efecto_variantes.tsv --database --species homo_sapiens --force_overwrite --tab'

Figura A8. UI para el panel de opciones del apartado 9 sin seleccionar opciones. Se pasa el ratón por encima de la opción del ensamblado del genoma para que se muestre el mensaje de explicación de la opción y se seleccionan dos campos para la tabla de resultados. Fuente: Elaboración propia.

Anexo 9. Apariencia de las opciones y el comando del apartado “9. Efecto de las variantes” tras seleccionar todas las opciones

GenAgans

The screenshot shows the GenAgans Shiny application interface. On the left, a sidebar lists navigation options: Inicio, 1. Control de calidad de las lecturas, 2. Limpieza de las lecturas, 3. Indexado del genoma de referencia, 4. Mapeo, 5. Análisis del mapeo, 6. Limpieza de duplicados, 7. Llamada de variantes, 8. Filtrado de variantes, 9. Efecto de las variantes (highlighted in teal), and Resumen.

The main panel is titled "Seleccione sus preferencias para la predicción del efecto de las variantes:". It contains several groups of checkboxes and dropdown menus:

- Ensamblado del genoma:** GRCh37
- Obtener significancia clínica:** En función del alelo
- Calcular SIFT:** Clasificación + puntuación
- Calcular PolyPhen:** Clasificación + Puntuación
- Obtener fenotipos asociados al gen afectado**
- Obtener la clase de variante (Sequence Ontology)**
- Obtener símbolo del gen afectado**
- Obtener el identificador Ensembl de la proteína**
- Obtener nomenclatura HGVS de la variante**
- Obtener nomenclatura SPDI de la variante**
- Obtener datos de variantes conocidas colocalizadas**
- Obtener referencias UniProt de proteínas colocalizadas**
- Obtener frecuencia alélica global de variantes colocalizadas**
- Limitar y ordenar los campos a incluir en la tabla:** Uploaded_variation x Location x

A text input field below the checkboxes asks: "Si desea añadir alguna opción adicional al comando, escríbala aquí:" with the value "-verbose".

A large text area at the bottom displays the command to be executed:

```
> vep -i ./8.Filtrado_variantes/variantes.vcf -o ./9.Efecto_variantes/efecto_variantes.tsv --database --species homo_sapiens --force_overwrite --tab --assembly GRCh37 --clin_sig_allele 1 --sift b --polyphen b --gene_phenotype --variant_class --symbol --protein --hgvs --spdi --check_existing --uniprot --af --verbose --fields "Uploaded_variation,Location"
```

Figura A9. UI para el panel de opciones y el comando del apartado 9 seleccionando todas las opciones. Fuente: Elaboración propia.

Anexo 10. Comparación entre la caja de resultados del apartado “2. Limpieza de las lecturas” contraída vs expandida

a

Comando que se ejecutará:

```
fastp -i SRR14695036_1.fastq.gz -I SRR14695036_2.fastq.gz -o
./2.Lecturas_limpias/SRR14695036_1_clean.fq.gz -O
./2.Lecturas_limpias/SRR14695036_2_clean.fq.gz --cut_tail 15 --
cut_front 15 --cut_mean_quality 30 --detect_adapter_for_pe --
trim_poly_g --trim_poly_x -l 100 -h
./2.Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.ht
ml -j
./2.Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.json
```

Iniciar limpieza de las lecturas

fastp report

Summary

General

fastp version: 0.23.4 (<https://github.com/OpenGene/fastp>)

Descargar resultados Siguiente

b

fastp report

Summary

General

fastp version:	0.23.4 (https://github.com/OpenGene/Fastp)
sequencing:	paired end (142 cycles + 142 cycles)
mean length before filtering:	138bp, 136bp
mean length after filtering:	137bp, 136bp
duplication rate:	24.370925%
Insert size peak:	142

Before filtering

Figura A10. UI para la caja de resultados del apartado 2. a) contraída y b) expandida. Fuente: Elaboración propia.

Anexo 11. Variación en el código servidor para la descarga de todos los archivos en el apartado “Resumen”

```
2861 @render.download
2862 def descarga.todos_archivos():
2863     total_archivos = sum([len(archivos) for raiz, subcarpetas, archivos in os.walk("./") if archivos != "Archivos_generados.zip"])
2864     with ui.Progress(min=1, max=total_archivos) as barra:
2865         barra.set(message="Procesando...", detail="Dependiendo de la cantidad de archivos generados este proceso puede\
2866             | ser más o menos pesado. Espere, por favor.")
2867         conteo_barra = 1
2868         barra.set(conteo_barra)
2869         with zipfile.ZipFile("Archivos_generados.zip", "w") as archivozip:
2870             for raiz, subcarpetas, archivos in os.walk(ruta_archivos):
2871                 for archivo in archivos:
2872                     if archivo != "Archivos_generados.zip":
2873                         ruta_archivo = os.path.join(raiz, archivo)
2874                         archivozip.write(ruta_archivo, os.path.relpath(ruta_archivo, "./"))
2875                         conteo_barra += 1
2876                         barra.set(conteo_barra)
2877         barra.set(total_archivos)
2878     return "Archivos_generados.zip"
```

Figura A11. Código en el servidor para la descarga de todos los archivos generados en el apartado “Resumen”.

Fuente: Elaboración propia.

Anexo 12. Desplegables del apartado “Resumen”

GenAgans

a



En este apartado puede descargar todos los archivos generados hasta el momento, visualizar un resumen de todos los resultados obtenidos, un resumen de todos los comandos ejecutados y descargar un script con ellos para lanzarlo desde la terminal si desea replicar exactamente el mismo flujo de trabajo de forma automática.

Tenga en cuenta que tanto los resultados como los comandos mostrados e incluidos en el script son los últimos que ha generado/ejecutado. Las opciones que tenga marcadas actualmente en los apartados no se reportarán aquí. De esta forma, se asegura la reproducibilidad del flujo de trabajo realizado.

Puede acceder a cada uno de estos subapartados a partir de los siguientes desplegables.

Resumen de los comandos ejecutados:

Script con los comandos ejecutados:

Resumen de resultados:

 Descargar todos los archivos generados

b



En este apartado puede descargar todos los archivos generados hasta el momento, visualizar un resumen de todos los resultados obtenidos, un resumen de todos los comandos ejecutados y descargar un script con ellos para lanzarlo desde la terminal si desea replicar exactamente el mismo flujo de trabajo de forma automática.

Tenga en cuenta que tanto los resultados como los comandos mostrados e incluidos en el script son los últimos que ha generado/ejecutado. Las opciones que tenga marcadas actualmente en los apartados no se reportarán aquí. De esta forma, se asegura la reproducibilidad del flujo de trabajo realizado.

Puede acceder a cada uno de estos subapartados a partir de los siguientes desplegables.

Resumen de los comandos ejecutados:

Script con los comandos ejecutados:

```
#!/bin/bash
mkdir 1.Analisis_calidad
```

 Descargar script (.sh)  Descargar script (.txt)

Resumen de resultados:

 Descargar todos los archivos generados

Figura A12. UI para los desplegables del apartado “Resumen”. a) Desplegables plegados y b) Desplegable del script personalizado desplegado. Fuente: Elaboración propia.

Anexo 13. Código del servidor para el script personalizado

Líneas de código: 2649 a 2793

```
@reactive.effect
  def actualizacion_script():
    comandos_ejecutados_html.set(f"#!/bin/bash <br>\  

                                <br>\n
                                {comando1a_ejecutado.get()} <br>\n
                                <br>\n
                                {comando1_ejecutado.get()} <br>\n
                                <br>\n
                                {comando2a_ejecutado.get()} <br>\n
                                <br>\n
                                {comando2b1_ejecutado.get()} <br>\n
                                <br>\n
                                {comando2b2_ejecutado.get()} <br>\n
                                <br>\n
                                {comando2_ejecutado.get()} <br>\n
                                <br>\n
                                {comando3a_ejecutado.get()} <br>\n
                                <br>\n
                                {comando3b_ejecutado.get()} <br>\n
                                <br>\n
                                {comando3_ejecutado.get()} <br>\n
                                <br>\n
                                {comando4a_ejecutado.get()} <br>\n
                                <br>\n
                                {comando4_ejecutado.get()} <br>\n
                                <br>\n
                                {comando5a_ejecutado.get()} <br>\n
                                <br>\n
                                {comando5b_ejecutado.get()} <br>\n
                                <br>\n
                                {comando5c_ejecutado.get()} <br>\n
                                <br>\n
                                {comando5_ejecutado.get()} <br>\n
                                <br>\n
                                {comando5d_ejecutado.get()} <br>\n
                                <br>\n
                                {comando6a_ejecutado.get()} <br>\n
                                <br>\n
                                {comando6b_ejecutado.get()} <br>\n
```

```
<br>\n{comando6c_ejecutado.get()} <br>\n<br>\n{comando6d0_ejecutado.get()} <br>\n<br>\n{comando6d_ejecutado.get()} <br>\n<br>\n{comando7a_ejecutado.get()} <br>\n<br>\n{comando7b_ejecutado.get()} <br>\n<br>\n{comando7c_ejecutado.get()} <br>\n<br>\n{comando7d_ejecutado.get()} <br>\n<br>\n{comando7e_ejecutado.get()} <br>\n<br>\n{comando7f_ejecutado.get()} <br>\n<br>\n{comando8a_ejecutado.get()} <br>\n<br>\n{comando8b0_ejecutado.get()} <br>\n<br>\n{comando8b_ejecutado.get()} <br>\n<br>\n{comando8_ejecutado.get()} <br>\n<br>\n{comando8c_ejecutado.get()} <br>\n<br>\n{comando8d_ejecutado.get()} <br>\n<br>\n{comando9a_ejecutado.get()} <br>\n<br>\n{comando9_ejecutado.get()}"\n\ncomandos_ejecutados_sh_txt.set(f"#!/bin/bash\n"\nf"\n"\nf"{comando1a_ejecutado.get()}\n"\nf"\n"\nf"{comando1_ejecutado.get()}\n"\nf"\n"\nf"{comando2a_ejecutado.get()}\n"\nf"\n"\nf"{comando2b1_ejecutado.get()}\n"\nf"\n"
```

```
f" {comando2b2_ejecutado.get()}\n"
f"\n"
f" {comando2_ejecutado.get()}\n"
f"\n"
f" {comando3a_ejecutado.get()}\n"
f"\n"
f" {comando3b_ejecutado.get()}\n"
f"\n"
f" {comando3_ejecutado.get()}\n"
f"\n"
f" {comando4a_ejecutado.get()}\n"
f"\n"
f" {comando4_ejecutado.get()}\n"
f"\n"
f" {comando5a_ejecutado.get()}\n"
f"\n"
f" {comando5b_ejecutado.get()}\n"
f"\n"
f" {comando5c_ejecutado.get()}\n"
f"\n"
f" {comando5_ejecutado.get()}\n"
f"\n"
f" {comando5d_ejecutado.get()}\n"
f"\n"
f" {comando6a_ejecutado.get()}\n"
f"\n"
f" {comando6b_ejecutado.get()}\n"
f"\n"
f" {comando6c_ejecutado.get()}\n"
f"\n"
f" {comando6d0_ejecutado.get()}\n"
f"\n"
f" {comando6d_ejecutado.get()}\n"
f"\n"
f" {comando7a_ejecutado.get()}\n"
f"\n"
f" {comando7b_ejecutado.get()}\n"
f"\n"
f" {comando7c_ejecutado.get()}\n"
f"\n"
f" {comando7d_ejecutado.get()}\n"
f"\n"
f" {comando7e_ejecutado.get()}\n"
f"\n"
f" {comando7f_ejecutado.get()}\n"
```

```
f"\n"
f"{comando8a_ejecutado.get()}\n"
f"\n"
f"{comando8b0_ejecutado.get()}\n"
f"\n"
f"{comando8b_ejecutado.get()}\n"
f"\n"
f"{comando8_ejecutado.get()}\n"
f"\n"
f"{comando8c_ejecutado.get()}\n"
f"\n"
f"{comando8d_ejecutado.get()}\n"
f"\n"
f"{comando9a_ejecutado.get()}\n"
f"\n"
f"{comando9_ejecutado.get()}")
```

Anexo 14. UI y prueba de funcionalidad de GenAgans

GenAgans

¡Bienvenido/a!
GenAgans es una aplicación que le permitirá realizar el flujo de trabajo de análisis de variantes genéticas en humanos a partir de lecturas pareadas de forma interactiva, intuitiva y personalizada.

Introducción de uso:

- Cargue los archivos de lecturas (formato FASTQ) y el genoma de referencia (formato FASTA).
- Los botones avance se irán activando conforme vaya completando los diferentes pasos. Síguenos.
- Visualice los resultados obtenidos para tomar decisiones sobre las opciones disponibles.
- En cualquier momento puede consultar el apartado de resumen y/o descargar los archivos generados.

1r archivo de lecturas
SRR14695036_1.fa
stq.gz

2o archivo de lecturas
SRR14695036_2.fa
stq.gz

Genoma de referencia
Homo_sapiens.GR
Ch37.dna.chromosome.1.fa

Cargue el primer archivo de lecturas
Cargar... SRR1469...
Upload complete

Cargue el segundo archivo de lecturas
Cargar... SRR1469...
Upload complete

Cargue el genoma de referencia para el mapeo
Cargar... Homo_sa...
Upload complete

EMPEZAR

Figura A14.1. UI para el apartado “Inicio” con los archivos cargados. Fuente: Elaboración propia.

GenAgans

Comando que se ejecutará:

```
> fastqc -o 1.Analisis_calidad/ SRR14695036_1.fastq.gz
SRR14695036_2.fastq.gz
```

Iniciar análisis de calidad de las lecturas

Basic Statistics

Basic Statistics

Descargar resultados

Siguiente

Figura A14.2. UI para el apartado 1 con los resultados contraídos. Fuente: Elaboración propia.

The screenshot shows a modal window titled "Basic Statistics" with a green checkmark icon. It displays a table of file metadata:

Measure	Value
Filename	SRR14695036_1.fastq.gz
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	15849308
Total Bases	2.1 Gbp
Sequences flagged as poor quality	0
Sequence length	50-142
%GC	49

Figura A14.3. UI para el apartado 1 con uno de los resultados expandidos. Fuente: Elaboración propia.

GenAgans

The screenshot shows the "Limpieza de las lecturas" (Cleaning) section of the UI. On the left, a sidebar lists steps: Inicio, Limpieza de las lecturas (selected), Indexado del genoma de referencia, Mapeo, Análisis del mapeo, Limpieza de duplicados, Llamada de variantes, Filtrado de variantes, Efecto de las variantes, and Resumen.

The main panel contains the following sections:

- Nota:** "Este apartado es OPCIONAL aunque altamente recomendable. Si realmente considera que sus archivos de lecturas no requieren ningún tipo de limpieza puede proceder directamente al siguiente apartado."
- Siguiente:** A button to proceed to the next step.
- Selección de preferencias:** Options for sequencing cleaning:
 - Restringir calidad media de las lecturas: Input field shows 30.
 - Recortar bases en el extremo 5' (inicio) de las lecturas: Input field shows 15.
 - Recortar bases en el extremo 3' (final) de las lecturas: Input field shows 15.
 - Restringir la longitud mínima de las lecturas: Input field shows 100.
- Opciones adicionales:** Checkboxes for: Eliminar adaptadores, Eliminar secuencias polyG, and Eliminar secuencias polyX.
- Comando adicional:** A text input field for adding extra command options.

Figura A14.4. UI para el apartado 2 con las opciones seleccionadas. Fuente: Elaboración propia.

Resumen

Si desea añadir alguna opción adicional al comando, escribeala aquí:

Comando que se ejecutará:

```
fastp -i SRR14695036_1.fastq.gz -I SRR14695036_2.fastq.gz -o
./Lecturas_limpias/SRR14695036_1_clean.fq.gz -O
./Lecturas_limpias/SRR14695036_2_clean.fq.gz --cut_tail 15 --
cut_front 15 --cut_mean_quality 30 --detect_adapter_for_pe --
trim_poly_g --trim_poly_x -l 100 -h
./Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.html -j
./Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.json
```

Iniciar limpieza de las lecturas

fastp report

Summary	
General	fastp version: 0.23.4 (https://github.com/OpenGene/fastp)
	Descargar resultados
	Siguiente

Figura A14.5. UI para el apartado 2 con las opciones seleccionadas y el resultado contraído. Fuente: Elaboración propia.

Summary	
General	fastp version: 0.23.4 (https://github.com/OpenGene/fastp)
sequencing:	paired end (142 cycles + 142 cycles)
mean length before filtering:	138bp, 136bp
mean length after filtering:	137bp, 136bp
duplication rate:	24.370925%
Insert size peak:	142

Before filtering

Figura A14.6. UI para el apartado 2 con el resultado expandido. Fuente: Elaboración propia.

GenAgans

Comando que se ejecutará:

```
bwa index -p
./3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.ch
romosome.1.fa
./3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.ch
romosome.1.fa
```

> Iniciar indexado del genoma de referencia

Descargar resultados | Siguiente

Figura A14.7. UI para el apartado 3 con los archivos cargados. Fuente: Elaboración propia.

GenAgans

Comando que se ejecutará:

```
bwa mem -a
./3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.ch
romosome.1.fa ./2.Lecturas_limpias/SRR14695036_1_clean.fq.gz
./2.Lecturas_limpias/SRR14695036_2_clean.fq.gz -o
./4.Mapeo/mapeo.sam
```

> Iniciar mapeo de las lecturas contra el genoma de referencia

Descargar resultados | Siguiente

Figura A14.8. UI para el apartado 4 con los archivos cargados. Fuente: Elaboración propia.

GenAgans

Inicio

1. Control de calidad de las lecturas
2. Limpieza de las lecturas
3. Indexado del genoma de referencia
4. Mapeo
- 5. Análisis del mapeo**
6. Limpieza de duplicados
7. Llamada de variantes
8. Filtrado de variantes
9. Efecto de las variantes

Resumen

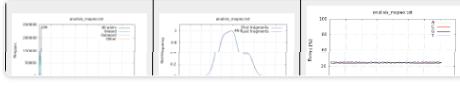
Primer comando que se ejecutará para la conversión del archivo SAM a BAM:
`samtools view -bS ./4.Mapeo/mapeo.sam -o ./5.Analisis_mapeo/mapeo.bam`

Segundo comando que se ejecutará para ordenar el archivo BAM por coordenadas:
`samtools sort ./5.Analisis_mapeo/mapeo.bam -o ./5.Analisis_mapeo/mapeo_sorted.bam`

Tercer comando que se ejecutará para obtener un informe sobre el resultado del mapeo:
`samtools stats ./5.Analisis_mapeo/mapeo_sorted.bam > ./5.Analisis_mapeo/analisis_mapeo.txt`

Cuarto comando que se ejecutará para graficar el informe sobre el resultado del mapeo:
`plot-bamstats -p ./5.Analisis_mapeo/informe_mapeo ./5.Analisis_mapeo/analisis_mapeo.txt`

Iniciar análisis del resultado del mapeo



Descargar resultados

Siguiente

Figura A14.9. UI para el apartado 5 con los resultados comprimidos. Fuente: Elaboración propia.

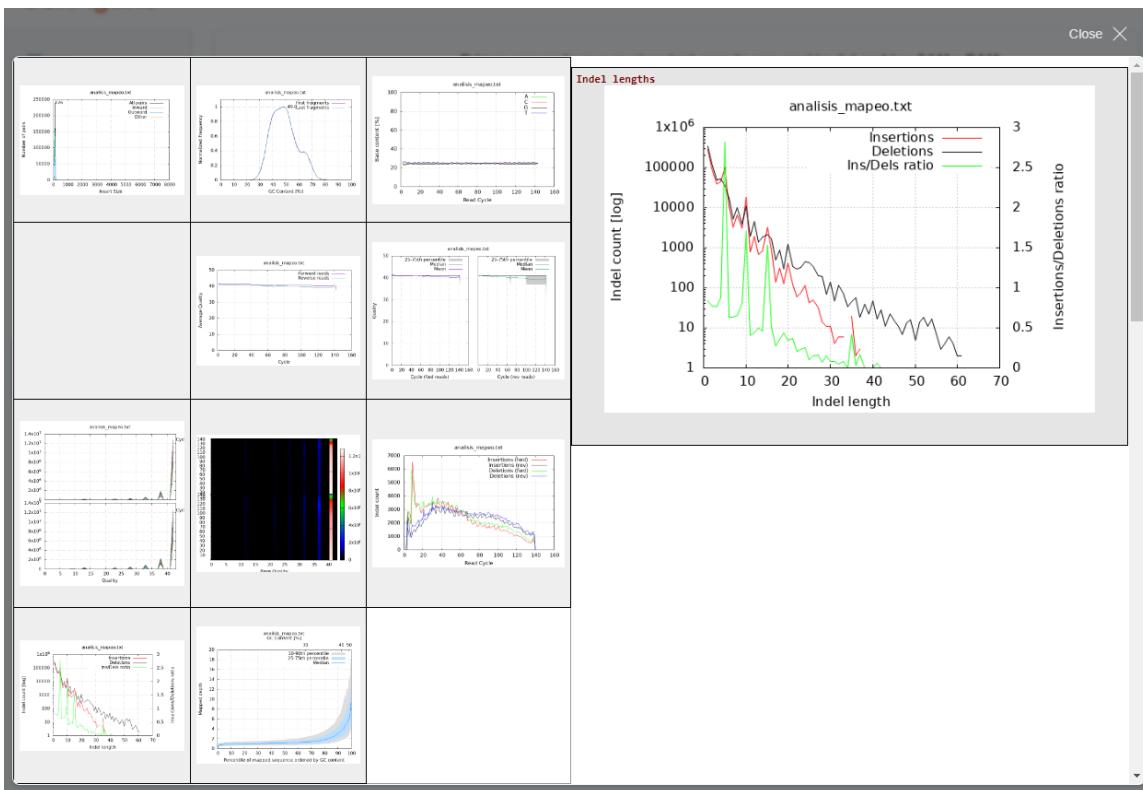


Figura A14.10. UI para el apartado 5 con los resultados expandidos. Fuente: Elaboración propia.

GenAgans

- ☒ Inicio**
- 1. Control de calidad de las lecturas
- 2. Limpieza de las lecturas
- 3. Indexado del genoma de referencia
- 4. Mapeo
- 5. Análisis del mapeo
- 6. Limpieza de duplicados**
- 7. Llamada de variantes
- 8. Filtrado de variantes
- 9. Efecto de las variantes
- ☒ Resumen**

Comando que se ejecutará para marcar los duplicados:

```
> picard MarkDuplicates --INPUT
./5.Analisis_mapeo/mapeo_sorted.bam --OUTPUT
./6.Limpieza_duplicados/mapeo_dedup.bam --METRICS_FILE
./6.Limpieza_duplicados/MarkDuplicatesMetrics.txt --
ASSUME_SORTED True
```

Iniciar marcaje de duplicados

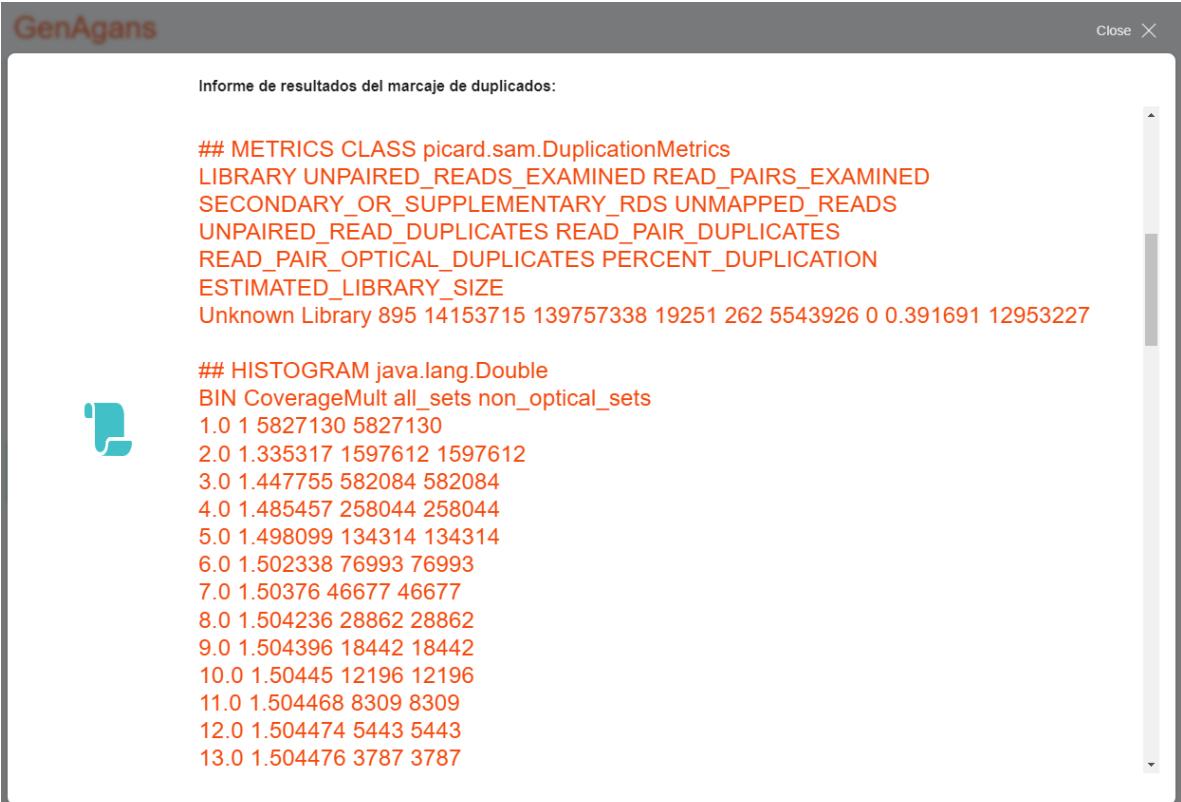
Informe de resultados del marcaje de duplicados:

```
## htsjdk.samtools.metrics.StringHeader
# MarkDuplicates --INPUT
./5.Analisis_mapeo/mapeo_sorted.bam --OUTPUT
./6.Limpieza_duplicados/mapeo_dedup.bam --METRICS_FILE
./6.Limpieza_duplicados/MarkDuplicatesMetrics.txt
```

Importante:

Los Read Groups (RG) son metadatos identificativos para las lecturas secuenciadas necesarios para realizar la llamada de variantes.
A continuación se realiza una comprobación de los RG presentes en el archivo de mapeo generado.
Los campos esperados son: "ID", "LB", "PL", "PU" y "SM". En función de lo observado en el informe deberá tomar una decisión:
- Si el informe aparece vacío, será necesaria su adición. Para ello emplee el último comando de este apartado.
- Si el informe muestra RG pero no para todos los campos, también será necesaria su adición. Para ello

Figura A14.11. UI para el apartado 6 con los resultados del marcaje de duplicados contraídos. Fuente: Elaboración propia.



The screenshot shows a window titled "GenAgans" with a "Close" button. Inside, there's a header "Informe de resultados del marcaje de duplicados:". Below it is a code block representing the output of a command. On the left side of the code block, there is a teal icon of a document with a blue exclamation mark.

```

## METRICS CLASS picard.sam.DuplicationMetrics
LIBRARY UNPAIRED_READS_EXAMINED READ_PAIRS_EXAMINED
SECONDARY_OR_SUPPLEMENTARY_RDS UNMAPPED_READS
UNPAIRED_READ_DUPLICATES READ_PAIR_DUPLICATES
READ_PAIR_OPTICAL_DUPLICATES PERCENT_DUPPLICATION
ESTIMATED_LIBRARY_SIZE
Unknown Library 895 14153715 139757338 19251 262 5543926 0 0.391691 12953227

## HISTOGRAM java.lang.Double
BIN CoverageMult all_sets non_optical_sets
1.0 1 5827130 5827130
2.0 1.335317 1597612 1597612
3.0 1.447755 582084 582084
4.0 1.485457 258044 258044
5.0 1.498099 134314 134314
6.0 1.502338 76993 76993
7.0 1.50376 46677 46677
8.0 1.504236 28862 28862
9.0 1.504396 18442 18442
10.0 1.50445 12196 12196
11.0 1.504468 8309 8309
12.0 1.504474 5443 5443
13.0 1.504476 3787 3787

```

Figura A14.12. UI para el apartado 6 con los resultados del marcaje de duplicados expandidos. Fuente: Elaboración propia.

- 7. Llamada de variantes
- 8. Filtrado de variantes
- 9. Efecto de las variantes
-  Resumen

Importante:

Los Read Groups (RG) son metadatos identificativos para las lecturas secuenciadas necesarios para realizar la llamada de variantes.

A continuación se realiza una comprobación de los RG presentes en el archivo de mapeo generado. Los campos esperados son: "ID", "LB", "PL", "PU" y "SM". En función de lo observado en el informe deberá tomar una decisión:

- Si el informe aparece vacío, será necesaria su adición. Para ello emplee el último comando de este apartado.
- Si el informe muestra RG pero no para todos los campos, también será necesaria su adición. Para ello emplee el último comando de este apartado (tenga en cuenta que se reemplazarán los valores actuales).
- Si el informe muestra RG para todos los campos, no será necesaria su adición. Puede ignorar el último comando de este apartado y seguir adelante.

Figura A14.13. Mensaje de advertencia sobre los RG en el apartado 6. Fuente: Elaboración propia.

The screenshot shows a Shiny application window. At the top left is the VIU logo. The main area contains several sections:

- Section 1:** A teal right-pointing arrow icon. Text: "Comando que se ejecutará para comprobar los grupos de lectura (Read Groups) actuales: samtools view -H ./6.Limpieza_duplicados/mapeo_dedup.bam | grep '^@RG' > ./6.Limpieza_duplicados/ReadGroups.txt".
- Section 2:** A teal right-pointing arrow icon. A button labeled "Iniciar comprobación de Read Groups".
- Section 3:** A teal right-pointing arrow icon. Text: "Informe de resultados de la comprobación de los Read Groups: @RG ID:default LB:lib1 PL:ILLUMINA SM:sample1 PU:unit1".
- Section 4:** A checkbox with a checked mark: "Si desea añadir Read Groups marque esta casilla, puede usar los valores predefinidos o introducir los propios". Below it are five input fields:
 - RGID - ID del grupo de lecturas (campo ID): default
 - RGLB - Nombre de la librería (campo LB): lib1
 - RGPL - Nombre del secuenciador (campo PL): ILLUMINA
 - RGSM - Nombre de la muestra (campo SM): sample1
 - RGPU - Nombre de la carrera (run) (campo PU): unit1
- Section 5:** A teal right-pointing arrow icon. Text: "Comando que se ejecutará para añadir los Read Groups: picard AddOrReplaceReadGroups -I ./6.Limpieza_duplicados/mapeo_dedup_copy.bam -O ./6.Limpieza_duplicados/mapeo_dedup.bam -RGID default -RGLB lib1 -RGPL ILLUMINA -RGSM sample1 -RGPU unit1".

Figura A14.14. UI para el apartado 6 con el comando de comprobación de RG ejecutado y el de adición o reemplazo de RG seleccionado. Fuente: Elaboración propia.

GenAgans

Inicio

1. Control de calidad de las lecturas
2. Limpieza de las lecturas
3. Indexado del genoma de referencia
4. Mapeo
5. Análisis del mapeo
6. Limpieza de duplicados
- 7. Llamada de variantes**
8. Filtrado de variantes
9. Efecto de las variantes

Resumen

Primer comando que se ejecutará para indexar el genoma de referencia:

```
> samtools faidx ./7.Llamada_variantes/Homo_sapiens.GRCh37.dna.chromosome.1.fa
```

Segundo comando que se ejecutará para indexar el archivo de mapeo:

```
> samtools index ./6.Limpieza_duplicados/mapeo_dedup.bam
```

Puede añadir opciones al comando para la llamada de variantes si lo desea:

<input checked="" type="checkbox"/> Ploidía La ploidía del organismo es de: 2	<input checked="" type="checkbox"/> Fracción mínima La fracción mínima de lecturas que apoyan el alelo alternativo para que se considere como una variante debe ser de: 0,3	<input checked="" type="checkbox"/> Cantidad de lecturas mínima La cantidad mínima de lecturas que apoyan el alelo alternativo para que se considere como una variante debe ser de: 100	<input checked="" type="checkbox"/> Calidad de mapeo mínima La calidad mínima de mapeo de una lectura necesaria para considerarla en la llamada variantes debe ser de: 30
---	---	---	---

Si desea añadir alguna opción adicional al comando, escribala aquí:

Tercer comando que se ejecutará para hacer la llamada de variantes:

```
> freebayes --ploidy 2 --min-alternate-fraction 0.3 --min-alternate-count 100 --min-mapping-quality 30 -f ./7.Llamada_variantes/Homo_sapiens.GRCh37.dna.chromosome.1.fa ./6.Limpieza_duplicados/mapeo_dedup.bam > ./7.Llamada_variantes/variantes.vcf
```

Cuarto comando que se ejecutará para obtener un informe de la llamada de variantes:

```
> rtg vcfstats ./7.Llamada_variantes/variantes.vcf > ./7.Llamada_variantes/informe_variantes.txt
```

Información de resultados de la llamada de variantes:

Location : ./7.Llamada_variantes/variantes.vcf
Failed Filters : 0
Passed Filters : 870
SNPs : 696

Opciones:

- Iniciar llamada de variantes
- Descargar resultados
- Siguiente

Figura A14.15. UI para el apartado 7 con los resultados contraídos. Fuente: Elaboración propia.

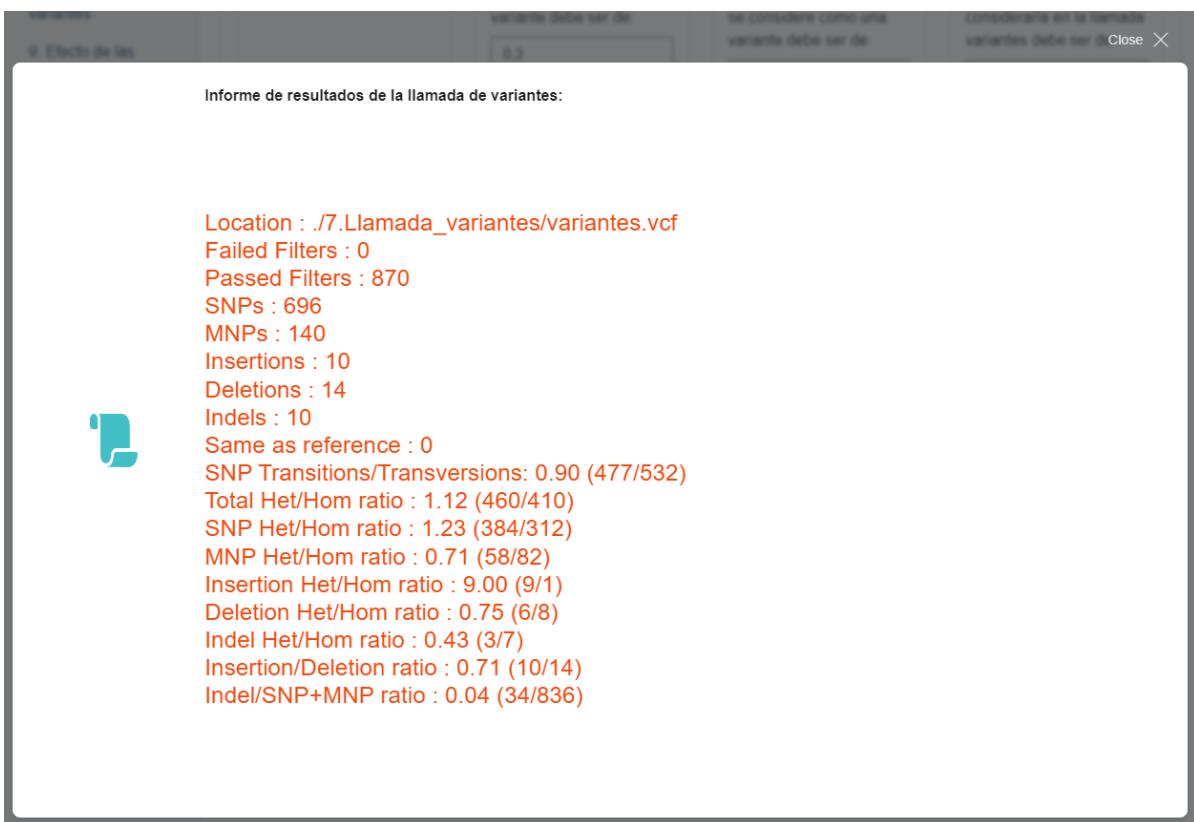


Figura A14.16. UI para el apartado 7 con los resultados expandidos. Fuente: Elaboración propia.

GenAgans

Este apartado es OPCIONAL
Si no desea filtrar su archivo de variantes puede proceder directamente al siguiente apartado.

Siguiente

Si desea filtrar las variantes, puede seleccionar alguna de estas opciones o añadir una propia:

Filtrar por calidad mínima Filtrar por profundidad mínima Filtrar por profundidad máxima

Eliminar variantes cuya calidad sea menor que:
30

Eliminar variantes cuya profundidad de cobertura sea menor que:
10

Eliminar variantes cuya profundidad de cobertura sea mayor que:
100

Eliminar los indels Mantener únicamente los indels

Si desea añadir alguna opción adicional al comando, escribala aquí:

Primer comando que se ejecutará para hacer el filtrado de variantes:

```
> vcftools --vcf ./7.Llamada_variantes/variantes_copy.vcf --recode --recode-INFO-all --out ./8.Filtrado_variantes/variantes.vcf --minQ 30 --minDP 10
```

Segundo comando que se ejecutará para obtener un informe del filtrado de variantes:

```
> rtg vcfstats ./8.Filtrado_variantes/variantes.vcf > ./8.Filtrado_variantes/informe_variantes_filtradas.txt
```

Iniciar filtrado de variantes

Informe de resultados del filtrado de variantes:

Location : ./8.Filtrado_variantes/variantes.vcf
Failed Filters : 0
Passed Filters : 870
SNPs : 696
.....

Descargar resultados Siguiente

Figura A14.17. UI para el apartado 8 con los resultados contraídos. Fuente: Elaboración propia.

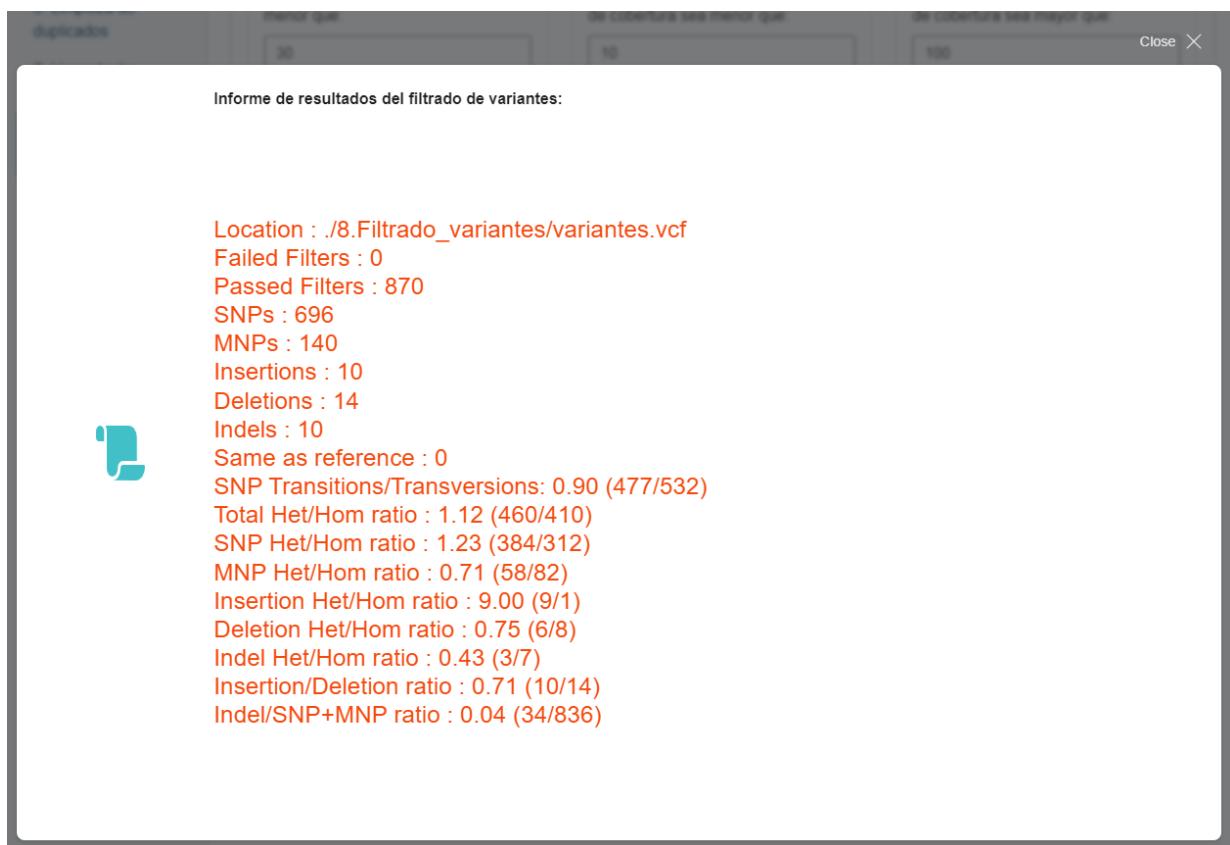


Figura A14.18. UI para el apartado 8 con los resultados expandidos. Fuente: Elaboración propia.

GenAgans

Inicio

1. Control de calidad de las lecturas
2. Limpieza de las lecturas
3. Indexado del genoma de referencia
4. Mapeo
5. Análisis del mapeo
6. Limpieza de duplicados
7. Llamada de variantes
8. Filtrado de variantes
- 9. Efecto de las variantes**

Resumen

Seleccione sus preferencias para la predicción del efecto de las variantes:

<input checked="" type="checkbox"/> Ensamblado del genoma: GRCh37	<input checked="" type="checkbox"/> Obtener significancia clínica: En función del alelo														
<input checked="" type="checkbox"/> Calcular SIFT Clasificación + puntuación	<input checked="" type="checkbox"/> Calcular PolyPhen Clasificación + Puntuación														
<input checked="" type="checkbox"/> Obtener fenotipos asociados al gen afectado	<input checked="" type="checkbox"/> Obtener la clase de variante (Sequence Ontology)														
<input checked="" type="checkbox"/> Obtener símbolo del gen afectado	<input checked="" type="checkbox"/> Obtener el identificador Ensembl de la proteína														
<input checked="" type="checkbox"/> Obtener nomenclatura HGVS de la variante	<input checked="" type="checkbox"/> Obtener nomenclatura SPDI de la variante														
<input checked="" type="checkbox"/> Obtener datos de variantes conocidas colocalizadas															
<input checked="" type="checkbox"/> Obtener referencias UniProt de proteínas colocalizadas															
<input checked="" type="checkbox"/> Obtener frecuencia alélica global de variantes colocalizadas															
<input type="checkbox"/> Limitar y ordenar los campos a incluir en la tabla:															
Si desea añadir alguna opción adicional al comando, escribala aquí:															
<pre>Comando que se ejecutará para predecir el efecto de las variantes: > vep -i ./8.Filtrado_variantes/variantes.vcf -o ./9.Efecto_variantes/efecto_variantes.tsv --database --species homo_sapiens --force_overwrite --tab --assembly GRCh37 -- clin_sig_allele 1 --sift b --polyphen b --gene_phenotype -- variant_class --symbol --protein --hgvs --spdi --check_existing -- uniprot --af</pre>															
<input type="button" value="Iniciar predicción del efecto de las variantes"/>															
Links <ul style="list-style-type: none"> Top of page VEP run statistics Data version General statistics 	VEP run statistics <p>VEP version (API) 112.0 (112)</p> <table border="1"> <tr> <td>Annotation sources</td> <td>homo_sapiens_core_112_37 on ensembl.ensembl.org</td> </tr> </table>	Annotation sources	homo_sapiens_core_112_37 on ensembl.ensembl.org												
Annotation sources	homo_sapiens_core_112_37 on ensembl.ensembl.org														
<table border="1"> <thead> <tr> <th>#Uploaded_variation</th> <th>Location</th> <th>Allele</th> <th>Gene</th> <th>Feature</th> <th>Feature_type</th> <th>Consequence</th> </tr> </thead> <tbody> <tr> <td>1_1647814_T/C</td> <td>1:1647814</td> <td>C</td> <td>ENSG000000000000</td> <td>ENST0000035620</td> <td>Transcript</td> <td>synonymous_variant</td> </tr> </tbody> </table>		#Uploaded_variation	Location	Allele	Gene	Feature	Feature_type	Consequence	1_1647814_T/C	1:1647814	C	ENSG000000000000	ENST0000035620	Transcript	synonymous_variant
#Uploaded_variation	Location	Allele	Gene	Feature	Feature_type	Consequence									
1_1647814_T/C	1:1647814	C	ENSG000000000000	ENST0000035620	Transcript	synonymous_variant									
<input type="button" value="Descargar resultados"/> <input type="button" value="Siguiente"/>															

Figura A14.19. UI para el apartado 9 con los resultados contraídos. Fuente: Elaboración propia.

#Uploaded_variation	Location	Allele	Gene	Feature	Feature_type	Consequence	cDNA_position	Protein_position	Amino_acids	Codon_change
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000356200	Transcript	synonymous_variant	592	-	-	3:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000356937	Transcript	non_coding_transcript_exon_variant	262	-	-	-
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000357760	Transcript	synonymous_variant	509	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000358779	Transcript	synonymous_variant	509	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000378633	Transcript	synonymous_variant	509	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000378635	Transcript	synonymous_variant	512	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000378638	Transcript	synonymous_variant	519	-	-	3:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000404249	Transcript	synonymous_variant	539	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000460465	Transcript	synonymous_variant,NMD transcript Variant	539	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000479362	Transcript	synonymous_variant	675	-	-	4:1
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000498810	Transcript	downstream_gene_variant	-	-	-	-
1_1647814_T/C	1:1647814	C	ENSG00000008128	ENST00000509982	Transcript	synonymous_variant,NMD transcript Variant	473	-	-	4:1
1_1647871_T/C	1:1647871	C	ENSG00000008128	ENST00000356200	Transcript	synonymous_variant	535	-	-	3:1

Figura A14.20. UI para el apartado 9 con los resultados de la tabla expandidos. #1. Fuente: Elaboración propia.

CDS_position	Protein_position	Amino_acids	Codons	Existing_variation	IMPACT	DISTANCE	STR
357	119	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
-	-	-	-	rs72901775,COSV62264481	MODIFIER	-	-1
429	143	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
429	143	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
429	143	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
429	143	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
357	119	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
459	153	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
459	153	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
429	143	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
-	-	-	-	rs72901775,COSV62264481	MODIFIER	2791	-1
459	153	E	gaA/gaG	rs72901775,COSV62264481	LOW	-	-1
300	100	R	cgA/cgG	rs72909014,COSV62264925	LOW	-	-1

Figura A14.21. UI para el apartado 9 con los resultados de la tabla expandidos. #2. Fuente: Elaboración propia.

STRAND	FLAGS	VARIANT_CLASS	SPDI	SYMBOL	SYMBOL_SOURCE	HGNC_ID	ENSP	SWISSP
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000348529	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	-	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000350403	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000351629	CD11A_HUMAN
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000367900	CD11A_HUMAN
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000367902	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000367905	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000384442	CD11A_HUMAN
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000462289	-
-1	cds_end_NF	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000423900	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	-	-
-1	-	SNV	NC_000001 .18:16478 13:T>C	CDK11A	HGNC	1730	ENSP00000422149	-
-1	-	SNV	NC_000001 .18:16478 78:T>C	CDK11A	HGNC	1730	ENSP00000348529	-

Figura A14.22. UI para el apartado 9 con los resultados de la tabla expandidos. #3. Fuente: Elaboración propia.

SWISSPROT	TREMBL	UNIPARC	UNIPROT_ISOFORM	GENE_PHENO	SIFT	PolyPhen	HGVSc	HG
-	Q5OPR3_HUMAN,Q4VBY6_HUMAN	UPI0000470903	-	-	-	-	ENST00000356200.3:c.357A>G	ENSP00000356200.3:c.357A>G
-	-	-	-	-	-	-	ENST00000356937.3:n.262A>G	-
-	Q5OPR3_HUMAN,Q50PO9_HUMAN,Q4VBV6_HUMAN,E9PFJ2_HUMAN	UPI0000456110	-	-	-	-	ENST00000357760.2:c.429A>G	ENSP00000357760.2:c.429A>G
CD11A_HUMAN	Q5OPQ9_HUMAN,Q4VBY6_HUMAN,E9PFJ2_HUMAN	UPI0000470904	-	-	-	-	ENST00000358779.5:c.429A>G	ENSP00000358779.5:c.429A>G
CD11A_HUMAN	Q5OPQ9_HUMAN,Q4VBY6_HUMAN,E9PFJ2_HUMAN	UPI0000366488	-	-	-	-	ENST00000378633.1:c.429A>G	ENSP00000378633.1:c.429A>G
-	Q5OPQ9_HUMAN,Q50PO9_HUMAN,Q4VBV6_HUMAN,E9PFJ2_HUMAN	UPI0002064DC1	-	-	-	-	ENST00000378635.3:c.429A>G	ENSP00000378635.3:c.429A>G
-	Q5QPR4_HUMAN,Q4VBY6_HUMAN	UPI0000470903	-	-	-	-	ENST00000378638.2:c.357A>G	ENSP00000378638.2:c.357A>G
CD11A_HUMAN	Q5QPR4_HUMAN,Q4VBY6_HUMAN	UPI0002281E1	-	-	-	-	ENST00000404249.3:c.459A>G	ENSP00000404249.3:c.459A>G
-	Q5QPR4_HUMAN,J3KS35_HUMAN	UPI000268AE0A	-	-	-	-	ENST00000460465.1:c.459A>G	ENSP00000460465.1:c.459A>G
-	Q5QPQ9_HUMAN,E9PFJ2_HUMAN	UPI0001D38C65	-	-	-	-	ENST00000479362.1:c.429A>G	ENSP00000479362.1:c.429A>G
-	Q5QPQ9_HUMAN,E7ESP2_HUMAN	UPI0001D38C55	-	-	-	-	ENST00000509882.1:c.459A>G	ENSP00000509882.1:c.459A>G
-	Q5QPR4_HUMAN,Q4VBY6_HUMAN	UPI0000470903	-	-	-	-	ENST00000356200.3:c.300A>G	ENSP00000356200.3:c.300A>G

Figura A14.23. UI para el apartado 9 con los resultados de la tabla expandidos. #4. Fuente: Elaboración propia.

GenAgans

The screenshot shows a table titled "GenAgans" with the following columns: GENE_PHENO, SIFT, PolyPhen, HGVSc, HGVSp, HGVS_OFFSET, AF, CLIN_SIG, SOMATIC, and PHENO. The table contains 10 rows of data, each representing a different genetic variant with its corresponding details.

GENE_PHENO	SIFT	PolyPhen	HGVSc	HGVSp	HGVS_OFFSET	AF	CLIN_SIG	SOMATIC	PHENO
-	-	-	ENST000003562 00_3:c.357A>G	ENSP000003485 00_3:c.357A>G	ENSP000003485 00_3:c.357A>G	-	-	-	0,1
-	-	-	ENST000003569 37_3:n..2624>G	-	-	-	-	-	0,1
-	-	-	ENST000003577 60_2:c.429A>G	ENSP000003504 03_2:p.Glu143>D	ENSP000003504 03_2:p.Glu143>D	-	-	-	0,1
-	-	-	ENST000003587 79_5:c.429A>G	ENSP000003516 29_5:p.Glu143>D	ENSP000003516 29_5:p.Glu143>D	-	-	-	0,1
-	-	-	ENST000003786 33_1:c.429A>G	ENSP000003679 00_1:p.Glu143>D	ENSP000003679 00_1:p.Glu143>D	-	-	-	0,1
-	-	-	ENST000003786 35_3:c.429A>G	ENSP000003679 02_3:p.Glu143>D	ENSP000003679 02_3:p.Glu143>D	-	-	-	0,1
-	-	-	ENST000003786 38_2:c.357A>G	ENSP000003679 05_1:p.Glu119>D	ENSP000003679 05_1:p.Glu119>D	-	-	-	0,1
-	-	-	ENST000004042 49_3:c.459A>G	ENSP000003844 42_3:p.Glu153>D	ENSP000003844 42_3:p.Glu153>D	-	-	-	0,1
-	-	-	ENST000004604 65_1:c.459A>G	ENSP000004622 69_1:p.Glu153>D	ENSP000004622 69_1:p.Glu153>D	-	-	-	0,1
-	-	-	ENST000004793 62_1:c.429A>G	ENSP000004239 00_1:p.Glu143>D	ENSP000004239 00_1:p.Glu143>D	-	-	-	0,1
-	-	-	ENST000005099 82_1:c.459A>G	ENSP000004221 49_1:p.Glu153>D	ENSP000004221 49_1:p.Glu153>D	-	-	-	0,1
-	-	-	ENST000003562 00_3:c.308A>G	ENSP000003485 29_2:p.Arg100>D	ENSP000003485 29_2:p.Arg100>D	-	-	-	0,1

Figura A14.24. UI para el apartado 9 con los resultados de la tabla expandidos. #5. Fuente: Elaboración propia.

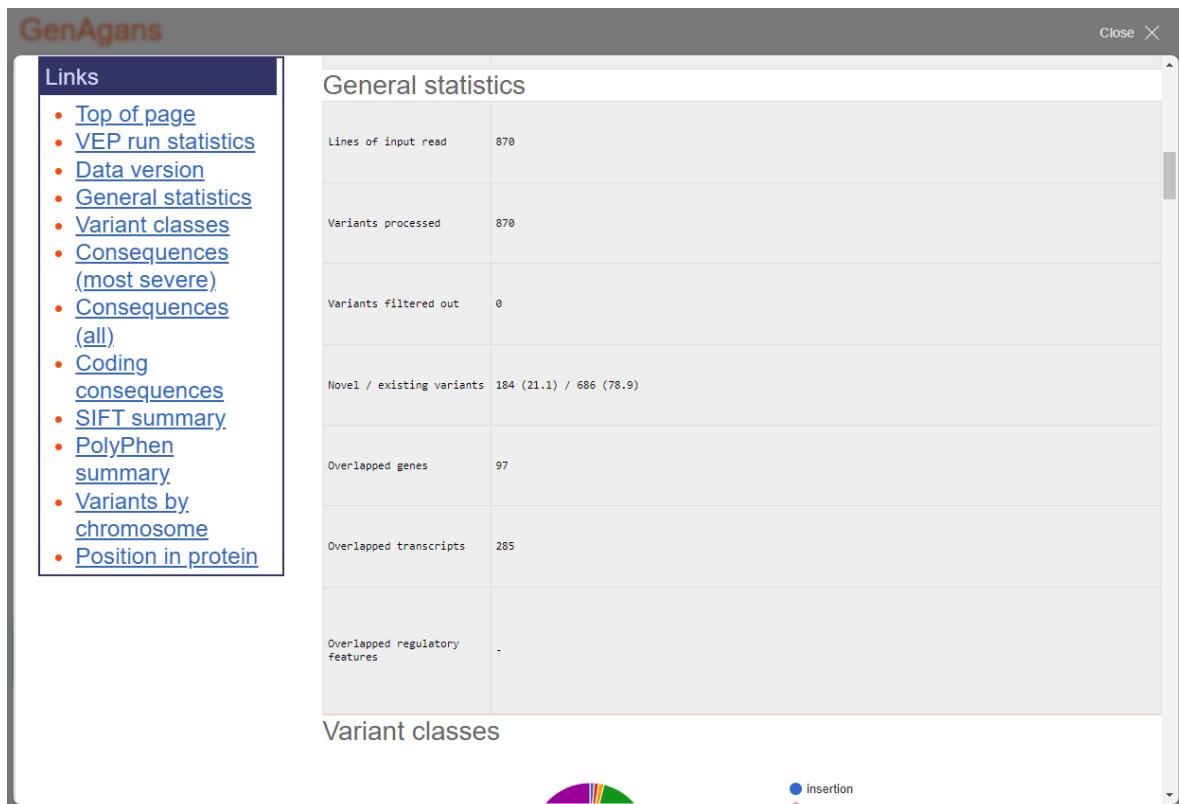


Figura A14.25. UI para el apartado 9 con los resultados generales expandidos. Fuente: Elaboración propia.

GenAgans

The screenshot shows the GenAgans application interface. On the left, a sidebar lists steps from 1 to 9. Step 9, 'Resumen', is highlighted with a teal button. The main area has a DNA icon and text explaining the 'Resumen' section. It states that users can download generated files, view a summary of results, and download a script to run it automatically. A note says that the commands shown are the last ones generated. Below this, a section titled 'Resumen de los comandos ejecutados:' is expanded, showing step 1 ('Control de calidad') with a command to generate a quality report for two fastq files. Step 2 ('Limpieza de lecturas') is also partially visible with its command.

Figura A14.26. UI del apartado “Resumen” con el resumen de comandos desplegado. Fuente: Elaboración propia.

GenAgans

This screenshot shows the same GenAgans interface as Figure A14.26, but with a different state. The 'Resumen' section is expanded, showing a 'Script con los comandos ejecutados:' block containing a '#!/bin/bash' shebang and a 'mkdir 1.Analisis_calidad' command. Below this are two download buttons: 'Descargar script (.sh)' and 'Descargar script (.txt)'. At the bottom, there's a 'Resumen de resultados:' section with a 'Descargar todos los archivos generados' button.

Figura A14.27. UI del apartado “Resumen” con el script personalizado desplegado. Fuente: Elaboración propia.

GenAgans

Close X

```

#!/bin/bash

mkdir 1.Analisis_calidad

fastqc -o 1.Analisis_calidad/ SRR14695036_1.fastq.gz SRR14695036_2.fastq.gz

mkdir 2.Lecturas_limpias

fastp -i SRR14695036_1.fastq.gz -I SRR14695036_2.fastq.gz -o ./2.Lecturas_limpias/SRR14695036_1_clean.fq.gz -O ./2.Lecturas_limpias/SRR14695036_2_clean.fq.gz --cut_tail 15 --cut_front 15 --cut_mean_quality 30 --detect_adapter_for_pe --trim_poly_g --trim_poly_x -l 100 -h ./2.Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.html -j ./2.Lecturas_limpias/SRR14695036_1_SRR14695036_2_fastp.json

mkdir 3.Genoma_referencia_indexado

cp Homo_sapiens.GRCh37.dna.chromosome.1.fa
./3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa

bwa index -p ./3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa
./3.Genoma_referencia_indexado/Homo_sapiens.GRCh37.dna.chromosome.1.fa

```

Figura A14.28. UI del apartado “Resumen” con el script personalizado expandido. Fuente: Elaboración propia.

GenAgans

1. Inicio

2. Control de calidad de las lecturas

3. Limpieza de las lecturas

4. Indexado del genoma de referencia

5. Mapeo

6. Análisis del mapeo

7. Limpieza de duplicados

8. Llamada de variantes

9. Filtrado de variantes

10. Resumen

En este apartado puede descargar todos los archivos generados hasta el momento, visualizar un resumen de todos los resultados obtenidos, un resumen de todos los comandos ejecutados y descargar un script con ellos para lanzarlo desde la terminal si desea replicar exactamente el mismo flujo de trabajo de forma automática.

Tenga en cuenta que tanto los resultados como los comandos mostrados e incluidos en el script son los últimos que ha generado/ejecutado. Las opciones que tenga marcadas actualmente en los apartados no se reportarán aquí. De esta forma, se asegura la reproducibilidad del flujo de trabajo realizado.

Puede acceder a cada uno de estos subapartados a partir de los siguientes desplegables.

Resumen de los comandos ejecutados:

Script con los comandos ejecutados:

Resumen de resultados:

Control de calidad:

Basic Statistics

Basic Statistics

Figura A14.29. UI del apartado “Resumen” con el resumen de resultados desplegado. Fuente: Elaboración propia.

Anexo 15. Medidas de control en la UI

GenAgans

Figura A15.1. UI del apartado “Inicio” sin archivos cargados. Fuente: Elaboración propia.

GenAgans

Figura A15.2. UI del apartado 1 sin archivos cargados. Fuente: Elaboración propia.

GenAgans

Comando que se ejecutará:

```
> fastqc -o 1.Analisis_calidad/ SRR14695036_1.fastq.gz  
SRR14695036_2.fastq.gz
```

Iniciar análisis de calidad de las lecturas

Informe de resultados del análisis de calidad del 1r archivo de lecturas:
Todavía no hay resultados

Informe de resultados del análisis de calidad del 2o archivo de lecturas:
Todavía no hay resultados

Descargar resultados

Siguiente

Figura A15.3. UI del apartado 1 con archivos cargados previa ejecución del comando. Fuente: Elaboración propia.