Bio-informatica groepswerk

Inhoudsopgave

1. Introductie tot Linux Shell/Bash

Wat is een Shell?

Een shell is een programma dat een interface biedt voor gebruikers om met het besturingssysteem te communiceren. De meest voorkomende shell in Linux-systemen heet Bash (Bourne Again SHell). Wanneer je de opdrachtregel gebruikt, typ je opdrachten in de shell.

Basisbegrippen

1. **Opdrachtprompt**: Hier typ je je opdrachten. Het eindigt meestal met een `$`-teken.
2. **Opdrachten**: Dit zijn instructies die je aan de computer geeft.
3. **Argumenten**: Aanvullende informatie die je aan een opdracht geeft.
4. **Opties**: Wijzigen het gedrag van opdrachten, meestal beginnend met een streepje (`-`).

Basisopdrachten

1. `pwd` (Print Working Directory)

Toont je huidige locatie in het bestandssysteem.

```
$ pwd
/home/gebruikersnaam
```

2. `ls` (List)

Geeft een lijst van bestanden en mappen in de huidige directory.

```
$ ls
Documenten  Downloads  Afbeeldingen  Muziek
```

Opties: - `ls -l`: Lang formaat, toont meer details - `ls -a`: Toont verborgen bestanden (die beginnen met een punt)

3. `cd` (Change Directory)

Verplaatst je naar een andere directory.

```
$ cd Documenten
```

Speciale directories: - `.` : Huidige directory - `..`: Bovenliggende directory - ~ : Thuisdirectory

4. `mkdir` (Make Directory)
Maakt een nieuwe directory aan.

```
$ mkdir NieuweMap
```

5. `cp` (Copy)
Kopieert bestanden of directories.

```
$ cp bestand.txt Documenten/
```

Om een directory en zijn inhoud te kopiëren, gebruik de `-r` (recursief) optie:

```
$ cp -r MapA MapB
```

6. `mv` (Move)
Verplaatst of hernoemt bestanden en directories.

```
$ mv bestand.txt Documenten/
$ mv oudenaam.txt nieuwenaam.txt
```

7. `rm` (Remove)
Verwijdert bestanden of directories. Wees voorzichtig met deze opdracht!

```
$ rm bestand.txt
```

Om een directory en zijn inhoud te verwijderen, gebruik de `-r` optie:

```
$ rm -r MapNaam
```

8. `cat` (Concatenate)
Toont de inhoud van een bestand.

```
$ cat bestand.txt
```

9. `echo`
Print tekst naar het scherm.

```
```

```
$ echo "Hallo, Wereld!"
Hallo, Wereld!
```

Opdrachtstructuur

De meeste opdrachten volgen deze structuur:

```
opdracht [opties] [argumenten]
```

Bijvoorbeeld:

```
$ ls -l Documenten
```

Hier is `ls` de opdracht, `-l` een optie, en `Documenten` een argument.

Tips

1. Gebruik de pijltjestoetsen omhoog en omlaag om door je opdrachtgeschiedenis te navigeren.
2. Gebruik Tab voor automatische aanvulling van bestands- en mapnamen.
3. Gebruik `man` gevolgd door een opdrachtnaam om de handleiding te zien (bijv. `man ls`).

Oefenopdrachten

1. Maak een directory genaamd “BioinformaticaCursus” in je thuisdirectory.
2. Maak binnen “BioinformaticaCursus” drie subdirectories: “Data”, “Scripts” en “Resultaten”.
3. Maak een leeg bestand genaamd “notities.txt” in de “BioinformaticaCursus” directory.
4. Toon de inhoud van “BioinformaticaCursus” in lang formaat.
5. Verplaats “notities.txt” naar de “Resultaten” directory.
6. Kopieer “notities.txt” van “Resultaten” naar “Data”.
7. Verwijder het “notities.txt” bestand uit de “Data” directory.

2. Kwaliteitscontrole met FastQC

Doel

Het hoofddoel van FastQC is om een snelle kwaliteitscontrole uit te voeren op ruwe sequentiedata afkomstig van high-throughput sequencing pijplijnen. Het helpt bij het identificeren van problemen die kunnen voortkomen uit de sequencer zelf of de bibliotheekvoorbereiding.

Opdrachten

```
# Voer FastQC uit op beide read-bestanden
fastqc monster1_R1.fastq.gz monster1_R2.fastq.gz
```

Het interpreteren van het FastQC HTML-rapport

1. **Basisstatistieken**: Geeft een overzicht van het bestand, waaronder totaal aantal sequenties, sequentielengte en GC-gehalte.
2. **Per base sequentiekwaliteit**: Toont de kwaliteitsscores voor elke positie in de read.

- Groen gebied: Goede kwaliteit
- Oranje gebied: Redelijke kwaliteit
- Rood gebied: Slechte kwaliteit

3. **Per tegelsequentiekwaliteit**: Relevant voor Illumina-sequencers, toont kwaliteit per tegel.
4. **Per sequentiekwaliteitsscores**: Geeft de verdeling van kwaliteitsscores over alle sequenties.
5. **Per base sequentie-inhoud**: Toont de verhoudingen van basen op elke positie.
6. **Per sequentie GC-inhoud**: Vergelijkt de waargenomen GC-inhoudverdeling met een theoretische normale verdeling.
7. **Per base N-inhoud**: Toont het percentage van basen op elke positie die niet konden worden bepaald (N).
8. **Sequentielengteverspreiding**: Voor de meeste platformen zou dit een scherpe piek moeten zijn.
9. **Sequentieduplicatieniveaus**: Hoge duplicatieniveaus kunnen duiden op PCR-bias.
10. **Overgerepresenteerde sequenties**: Lijst van sequenties die vaker voorkomen dan verwacht.
11. **Adapter-inhoud**: Toont de aanwezigheid van vaak gebruikte adapters in je bibliotheek.

Opdrachtvragen:
1. Wat is de gemiddelde kwaliteitsscore over alle basen?
2. Zijn er overgerepresenteerde sequenties? Zo ja, wat zouden deze kunnen zijn?
3. Hoe verandert de kwaliteitsscore over de lengte van de reads?

3. Read Mapping met BWA

BWA (Burrows-Wheeler Aligner) wordt gebruikt om de reads uit te lijnen tegen een referentiegenoom.

```
# Indexeer het referentiegenoom
bwa index referentiegenoom.fasta

# Lijn reads uit tegen het referentiegenoom
bwa mem referentiegenoom.fasta monster1_R1.fastq.gz monster1_R2.fastq.gz >
monster1.sam
```

4. Variant Calling met BCFtools

BCFtools wordt gebruikt om varianten te identificeren uit de uitgelijnde reads.

```
# Roep varianten aan
bcftools mpileup -f referentiegenoom.fasta monster1.sorted.bam | bcftools call
-mv -Ob -o monster1.raw.bcf
```

5. Variant Filtering

VCFtools kan worden gebruikt om de varianten te filteren op basis van verschillende criteria.

```
# Zet BCF om naar VCF
bcftools view monster1.raw.bcf > monster1.raw.vcf

# Filter varianten
vcftools --vcf monster1.raw.vcf \
         --minQ 30 \
         --min-meanDP 10 \
         --max-missing 0.8 \
         --recode --recode-INFO-all \
         --out monster1.gefilterd
```

6. Variant Annotatie met SnpEff

Stap 1: Installeer SnpEff

```
wget https://snpeff.blob.core.windows.net/versions/snpEff_latest_core.zip
unzip snpEff_latest_core.zip
```

Stap 2: Download de Human Genome Database

```
java -jar snpEff.jar download -v hg38
```

Stap 3: Voer SnpEff uit

```
java -Xmx4g -jar snpEff.jar hg38 input_varianten.vcf >
geannoteerde_varianten.vcf
```

Stap 4: Interpreteer de Resultaten

Kijk naar het `ANN` veld in de output VCF voor gedetailleerde annotaties.

Opdrachtvragen

1. Hoeveel varianten met HOGE impact heb je gevonden? Wat voor soort varianten zijn dit?
2. Vind een missense variant. Wat is de aminozuurverandering? In welk gen kwam het voor?
3. Zijn er varianten in bekende ziekte-geassocieerde genen?
4. Wat is het meest voorkomende type variant in je dataset?
5. Kun je varianten vinden die de eiwitfunctie kunnen beïnvloeden? Leg uit waarom je denkt dat ze impactvol kunnen zijn.

7. Simuleren van FASTQ-bestanden met NEAT

Stap 1: Incorporeer Varianten in Referentiegenoom

```
# Installeer NEAT (indien nog niet geïnstalleerd)
git clone https://github.com/ncbi/NEAT.git
cd NEAT
make

# Gebruik NEAT om varianten te incorporeren
./neat-genreads \
  -rc referentiegenoom.fa \
  -dv varianten.vcf \
  -o gemuteerd_genoom
```

Stap 2: Genereer FASTQ-bestanden

```
# Gebruik NEAT om FASTQ-bestanden te genereren
neat-genreads \
  -rf gemuteerd_genoom.fa \
  -R 150 \
  -c 30 \
  -o gesimuleerde_reads
```

Dit zal paired-end reads genereren met een lengte van 150 basen en 30x dekking.

Python Script voor Meerdere Monsters

Hier is een Python-script om FASTQ-bestanden voor meerdere monsters te genereren, elk met hun eigen set varianten:

```
import os
import subprocess
import argparse

def run_neat(reference, vcf, output_prefix, read_length, coverage):
    command = [
        "neat-genreads",
        "-rc", reference,
        "-dv", vcf,
        "-o", output_prefix,
        "-rl", str(read_length),
        "-c", str(coverage),
        "-pe", "150", "50",  # Paired-end reads, 150bp lengte, 50bp stddev
        "-bq", "30"  # Basiskwaliteitsscore
    ]
    subprocess.run(command, check=True)

def main():
    parser = argparse.ArgumentParser(description="Genereer FASTQ-bestanden voor
meerdere monsters met NEAT.")
   parser.add_argument("reference", help="Pad naar het referentiegenoombestand")
```

```
    parser.add_argument("vcf_dir", help="Directory met VCF-bestanden")
       parser.add_argument("output_dir",  help="Directory  voor  output  FASTQ-
bestanden")
              parser.add_argument("--read_length",    type=int,    default=150,
help="Leeslengte")
                 parser.add_argument("--coverage",    type=int,    default=30,
help="Dekkingsdiepte")

    args = parser.parse_args()

    os.makedirs(args.output_dir, exist_ok=True)

    for vcf_file in os.listdir(args.vcf_dir):
        if vcf_file.endswith(".vcf"):
            sample_name = os.path.splitext(vcf_file)[0]
            vcf_path = os.path.join(args.vcf_dir, vcf_file)
            output_prefix = os.path.join(args.output_dir, sample_name)

            print(f"FASTQ genereren voor monster: {sample_name}")
            run_neat(args.reference, vcf_path, output_prefix, args.read_length,
args.coverage)

    print("Alle monsters zijn succesvol verwerkt!")

if __name__ == "__main__":
    main()
```

Gebruik dit script als volgt:

```
python genereer_multi_monster_fastq.py /pad/naar/referentie.fa /pad/naar/
vcf_directory /pad/naar/output_directory
```

Conclusie

Deze cursus biedt een uitgebreide introductie tot bioinformatica, van basiscommando's in Linux tot geavanceerde onderwerpen zoals variant-annotatie en het simuleren van sequentiedata. Door deze stappen te volgen en de opdrachten uit te voeren, zul je een solide basis leggen voor verder werk in bioinformatica.