Semana 1: El Entorno del Ingeniero

Linux, Shell Scripting y Control de Versiones

Curso IA Agroindustria 2026

Enero 2026

Resumen

El 80 % del éxito de un proyecto de Inteligencia Artificial depende de la ingeniería de datos, automatización y reproducibilidad. Esta semana aprenderás las herramientas fundamentales: terminal Linux, procesamiento de datos con pipes y control de versiones con Git.

Índice

1. Fase 1: Supervivencia en la Terminal **4**
- 1.1. Navegación Básica . . . 4
- 1.2. Gestión de Archivos . . . 5
- 1.3. Comandos Clave de Científico de Datos . . . 6

2. Fase 2: Procesamiento y Automatización **8**
- 2.1. Inspección de Datos . . . 8
- 2.2. Datos Reales y Errores Comunes . . . 8
- 2.3. Pipelines con Pipes . . . 9
- 2.4. Procesamiento con awk . . . 9

3. Fase 3: Control de Versiones con Git **10**
- 3.1. Git como Bitácora Científica . . . 10
- 3.2. Uso de .gitignore . . . 10
- 3.3. Flujo de Trabajo Básico . . . 11

4. Casos de Uso por Carrera **13**
- 4.1. Agroindustria Alimenticia . . . 13
- 4.2. Agronomía . . . 13
- 4.3. Ingeniería Agrícola . . . 13
- 4.4. Ingeniería Ambiental . . . 14

5. Entregable A1: Script de Setup Reproducible **15**
- 5.1. Requisitos Mínimos . . . 15
- 5.2. Estructura Esperada del Repositorio . . . 15
- 5.3. Plantilla del Script . . . 15
- 5.4. Entrega . . . 16

Objetivos de Aprendizaje

Al finalizar esta semana podrás:

- Navegar y manipular archivos desde la terminal Linux sin GUI.
- Construir pipelines de procesamiento de datos con pipes y `awk`.
- Usar Git como bitácora científica de tus experimentos.
- Automatizar tareas repetitivas con scripts Bash reproducibles.

Tiempo estimado

En clase (4h): Fases 1–3 (terminal + Git básico).
Autoestudio: Fases 4–5 (estructura de proyectos + Python desde terminal).

Prefacio: Tu Laboratorio Digital

Imagina una finca equipada con sensores que generan miles de registros diarios en archivos CSV comprimidos. Abrir archivos manualmente no es una opción sostenible cuando los datos crecen. La ciencia de datos comienza cuando automatizas, validas y documentas cada paso del flujo de trabajo.

Principio Fundamental

Un modelo que no puede reproducirse no es ciencia: es magia.

1. Fase 1: Supervivencia en la Terminal

La terminal es un lenguaje de programación declarativo para hablar con el sistema operativo. Todo flujo de datos sigue la lógica:

$$\text{entrada} \rightarrow \text{proceso} \rightarrow \text{salida}$$

En esta fase aprenderás a orientarte en el sistema de archivos y manipular archivos como un científico de datos.

1.1. Navegación Básica

Objetivo

Aprender a saber dónde estás, qué archivos hay y cómo moverte entre carpetas de un proyecto de datos.

Comandos clave:

- `pwd`: muestra la ruta completa de la carpeta actual.
- `ls -lh`: lista archivos con tamaños legibles.
- `ls *.csv`: lista solo archivos CSV en la carpeta actual.
- `cd carpeta`: cambia a una subcarpeta.
- `cd ..`: sube un nivel en la jerarquía.
- `cd ~`: va a la carpeta personal del usuario.

```
# Ver en qué carpeta estás
pwd
/workspaces/curso-ia-agroindustria-2

# Listar el contenido de la carpeta actual
ls -lh
total 28K
-rw-rw-rw-  1 vscode root 1.1K Jan 19 03:22 LICENSE
-rw-rw-rw-  1 vscode root 4.7K Jan 19 03:22 README.md
drwxrwxrwx+ 5 vscode root 4.0K Jan 19 03:22 docs
drwxrwxrwx+ 2 vscode root 4.0K Jan 19 03:22 reports
drwxrwxrwx+ 2 vscode root 4.0K Jan 19 03:22 scripts
drwxrwxrwx+ 4 vscode root 4.0K Jan 19 03:22 talleres

# Para crear los archivos con datos ejecuta el siguiente script:
./scripts/setup_proyecto.sh

# Se creó la carpeta data, así que vamos a listar solo archivos CSV
ls data/raw/

datos_sensores.csv
```

Ejercicio guiado (15 min)

1. Crea una carpeta `proyecto_ia` dentro de tu `~`.
2. Dentro de `proyecto_ia`, crea las carpetas `data`, `scripts` y `reports`.
3. Usa `pwd` y `ls -lh` en cada paso para verificar que estás en la carpeta correcta.

1.2. Gestión de Archivos

Objetivo

Crear, copiar, mover y eliminar archivos y carpetas para organizar un proyecto de datos.

Comandos frecuentes:

- `mkdir -p ruta`: crea una carpeta (y las intermedias si no existen).
- `touch archivo`: crea un archivo vacío o actualiza su fecha.
- `cp origen destino`: copia archivos.
- `mv origen destino`: mueve o renombra archivos.
- `rm archivo`: elimina archivos.

```
# Crear estructura de carpetas del proyecto
mkdir -p proyecto_ia/data/raw
mkdir -p proyecto_ia/data/processed
mkdir -p proyecto_ia/scripts
mkdir -p proyecto_ia/reports

# Crear un archivo de notas vacío
cd proyecto_ia
touch notas_proyecto.md

# Copiar un CSV de sensores al directorio raw
cp ~/sensores.csv data/raw/sensores.csv

# Renombrar el archivo de notas
mv notas_proyecto.md README.md

# Eliminar un archivo (con confirmación si usas alias de seguridad)
rm data/raw/archivo_innecesario.csv
```

Advertencia

`rm` elimina permanentemente. Usa siempre `rm -i` para que pregunte antes de borrar cada archivo. Más adelante definiremos alias de seguridad.

Ejercicio (20 min)

Diseña la estructura de carpetas para un proyecto de monitoreo de humedad de suelo y crea:

- Una carpeta `figures` para gráficos.
- Una carpeta `logs` para registros de ejecución.
- Un archivo `TODO.md` en la raíz del proyecto.

1.3. Comandos Clave de Científico de Datos

Objetivo

Medir uso de disco, buscar archivos de datos y operar sobre columnas de archivos CSV desde la terminal.

Comandos útiles:

- `du -sh *`: muestra el tamaño de cada archivo/carpeta.
- `df -h`: muestra el espacio disponible en discos.
- `find . -name "*.csv"`: busca archivos con extensión `.csv`.
- `cut -d',' -f1`: extrae columnas de un CSV.
- `sort | uniq`: ordena y elimina duplicados.

```
# Ver qué carpeta ocupa más espacio dentro de data
cd proyecto_ia/data
du -sh *

# Ver espacio disponible en el disco
df -h

# Buscar todos los archivos CSV en el proyecto
cd ..
find . -name "*.csv"

# Extraer la primera columna (por ejemplo, id_sensor) de un CSV
cd data/raw
cut -d',' -f1 sensores.csv | head

# Obtener lista única de sensores
cut -d',' -f1 sensores.csv | sort | uniq | head
```

Ejercicio aplicado (25 min)

Supón que `sensores.csv` tiene columnas `id_sensor,fecha,temperatura,humedad`.

1. Obtén la lista única de `id_sensor` ordenada alfabéticamente.
2. Cuenta cuántos registros totales tiene el archivo.
3. Calcula cuántos registros hay por cada `id_sensor` usando `sort | uniq -c`.

2. Fase 2: Procesamiento y Automatización

En esta fase aprenderás a inspeccionar datos reales, detectar errores frecuentes y construir pequeños pipelines de procesamiento usando la terminal.

2.1. Inspección de Datos

Objetivo

Inspeccionar rápidamente el contenido de archivos grandes sin abrirlos en un editor gráfico.

Comandos:

- `head -n 5 datos.csv`: muestra las primeras 5 líneas.
- `tail -n 5 datos.csv`: muestra las últimas 5 líneas.
- `wc -l datos.csv`: cuenta las líneas del archivo.
- `less datos.csv`: permite navegar por el archivo.

```
# Ver las primeras líneas de sensores.csv
cd proyecto_ia/data/raw
head -n 5 sensores.csv

# Ver las últimas líneas (útil para ver fechas recientes)
tail -n 5 sensores.csv

# Contar cuántos registros hay
wc -l sensores.csv

# Navegar por el archivo (q para salir)
less sensores.csv
```

2.2. Datos Reales y Errores Comunes

Los datos reales contienen errores:

- Valores faltantes representados como cadenas vacías o símbolos especiales.
- Separadores inconsistentes (comas, punto y coma, tabuladores).
- Registros corruptos con caracteres no ASCII.

```
# Buscar columnas vacías consecutivas ,, que pueden indicar datos
    faltantes
grep ",," sensores.csv | head

# Detectar caracteres no ASCII (posibles errores de codificación)
grep -P "[^\x00-\x7F]" sensores.csv | head
```

Atención

Antes de entrenar cualquier modelo, debes conocer la calidad de tus datos. No confíes en que el archivo está limpio solo porque se abre en una hoja de cálculo.

2.3. Pipelines con Pipes

El operador | (pipe) conecta la salida de un comando con la entrada de otro. Permite encadenar operaciones simples para construir un procesamiento más complejo.

```
# Contar cuántas veces aparece cada id_sensor en el archivo
cut -d',' -f1 sensores.csv | sort | uniq -c | sort -nr | head

# Obtener los 10 valores de temperatura más altos registrados
cut -d',' -f3 sensores.csv | sort -nr | head

# Filtrar solo las líneas de un sensor específico y contarlas
grep "SENSOR_05" sensores.csv | wc -l
```

Ejercicio de pipeline (30 min)

Construye un pipeline que:

1. Se quede solo con las columnas `id_sensor,temperatura`.
2. Ordene por temperatura descendente.
3. Muestre las 5 mediciones más altas por pantalla.

2.4. Procesamiento con awk

`awk` es un mini-lenguaje para procesar texto y columnas; permite calcular estadísticas básicas directamente desde la terminal.

```
# Calcular la temperatura promedio (suponiendo que está en la columna 3)
awk -F',' '{s+=$3} END {print "Temperatura promedio:", s/NR}' sensores.
    csv

# Calcular la humedad máxima (columna 4)
awk -F',' 'NR>1 {if($4>max) max=$4} END {print "Humedad máxima:", max}'
    sensores.csv

# Filtrar solo filas con humedad menor a 30
awk -F',' '$4 < 30 {print $0}' sensores.csv | head
```

Ejercicio con awk (30 min)

1. Usa `awk` para calcular la temperatura mínima.
2. Calcula la suma total de registros donde `humedad < 40`.
3. Guarda las filas con `humedad > 80` en un archivo `alertas_humedad.csv`.

3. Fase 3: Control de Versiones con Git

En esta fase transformarás tu carpeta de archivos sueltos en un proyecto versionado con Git, usando buenas prácticas de colaboración.

3.1. Git como Bitácora Científica

Git es un sistema de control de versiones distribuido que registra la historia de tu proyecto, permitiéndote volver a estados anteriores y colaborar con otros.

Flujo básico:

1. `git init`: inicializa el repositorio.
2. `git status`: muestra el estado actual.
3. `git add`: selecciona cambios para guardar.
4. `git commit`: guarda un ”fotogramaçon mensaje.

```
cd ~/proyecto_ia

# Inicializar repositorio Git
git init

# Ver estado (archivos sin seguimiento)
git status

# Añadir todos los archivos para el primer commit
git add .

# Crear el commit inicial con un mensaje descriptivo
git commit -m "feat: estructura inicial del proyecto de sensores"

# Ver el historial de commits
git log --oneline
```

Buenas prácticas de commits

- Haz commits pequeños y frecuentes.
- Usa mensajes que expliquen el *por qué* del cambio.
- Agrupa cambios relacionados en un mismo commit.
- Usa prefijos: `feat:`, `fix:`, `docs:`, `refactor:`.

3.2. Uso de .gitignore

El archivo `.gitignore` indica a Git qué archivos o carpetas no deben versionarse (datos brutos, modelos grandes, archivos temporales).

```
# Crear el archivo .gitignore
cat > .gitignore << EOF
# Entornos virtuales
```

```
venv/
ia_env/

# Datos brutos (no versionables)
data/raw/
data/interim/

# Artefactos Python
*.pyc
__pycache__/
.ipynb_checkpoints/

# Modelos pesados
*.pth
*.h5

# Logs y temporales
*.log
*.tmp
EOF

git add .gitignore
git commit -m "docs: configura .gitignore para datos y artefactos"
```

Regla práctica

Nunca subas datos sensibles ni archivos de gran tamaño al repositorio. Para datos grandes, considera herramientas como Git LFS u otros almacenes externos.

3.3. Flujo de Trabajo Básico

```
# Modificar un archivo (por ejemplo, agregar un script)
echo "#!/bin/bash" > scripts/validar_datos.sh
echo "wc -l data/raw/*.csv" >> scripts/validar_datos.sh
chmod +x scripts/validar_datos.sh

# Ver qué cambió
git status
git diff scripts/validar_datos.sh

# Registrar el cambio
git add scripts/validar_datos.sh
git commit -m "feat: script de validación de registros"

# Ver historial
git log --oneline --graph
```

Ejercicio de Git (30 min)

1. Inicializa un repositorio en `proyecto_ia`.
2. Haz un commit inicial con la estructura de carpetas.
3. Crea un script `scripts/resumen.sh` que cuente archivos CSV y haz commit.
4. Modifica el README y haz otro commit con mensaje descriptivo.
5. Usa `git log` para ver tu historial.

4. Casos de Uso por Carrera

Los comandos aprendidos se aplican distinto según el dominio:

4.1. Agroindustria Alimenticia

Caso: Control de calidad de lotes

Archivos CSV con registros de inspección visual (color, tamaño, defectos) de fruta para exportación. Cada fila = 1 unidad inspeccionada, con columnas: `lote_id,fecha,peso_g,calibre,color,defectos`.

Pipeline típico:

```
# Cuántos lotes inspeccionados hoy
grep "$(date +%Y-%m-%d)" inspeccion.csv | cut -d',' -f1 | sort |
    uniq | wc -l

# Lotes con >5% de defectos (columna 6 > 5)
awk -F',' '$6 > 5 {print $1}' inspeccion.csv | sort | uniq >
    lotes_rechazados.txt
```

4.2. Agronomía

Caso: Monitoreo de rendimiento por lote

CSV con datos de cosecha: `lote_id,ha,kg_cosechados,fecha`.

Pipeline típico:

```
# Calcular toneladas/ha promedio por lote
awk -F',' 'NR>1 {print $1, $3/$2/1000}' cosecha.csv | \
  awk '{s[$1]+=$2; c[$1]++} END {for(l in s) print l, s[l]/c[l]}' |
      \
  sort -k2 -nr > rendimiento_promedio.txt
```

4.3. Ingeniería Agrícola

Caso: Telemetría de riego

Datos de sensores de humedad cada 15 min: `sensor_id,timestamp,humedad_suelo,presion_agua`.

Pipeline típico:

```
# Detectar sensores con humedad <20% (alerta riego)
awk -F',' '$3 < 20 {print $1,$2,$3}' riego.csv | \
  sort | uniq > alertas_riego.txt

# Contar alertas por sensor
cut -d' ' -f1 alertas_riego.txt | sort | uniq -c | sort -nr
```

4.4. Ingeniería Ambiental

Caso: Monitoreo de calidad de agua

CSV con muestreos: `estacion_id,fecha,pH,turbidez_NTU,coliformes_UFC`.
Pipeline típico:

```
# Estaciones fuera de norma (pH <6.5 o >8.5)
awk -F',' '$3<6.5 || $3>8.5 {print $1,$2,$3}' calidad_agua.csv >
    pH_fuera_norma.txt

# Promedios de turbidez por estación
awk -F',' 'NR>1 {s[$1]+=$4; c[$1]++} END {for(e in s) print e, s[e]/
    c[e]}' \
  calidad_agua.csv | sort -k2 -nr
```

Ejercicio multi-carrera (30 min)

Adapta uno de estos pipelines a tu dataset real (o simula uno pequeño con 20 filas en Excel y exporta a CSV). Documenta en tu README qué hizo cada comando y qué insights descubriste.

5. Entregable A1: Script de Setup Reproducible

Objetivo

Construir un script Bash (`scripts/setup_proyecto.sh`) que automatice la creación de estructura, descarga de datos (muestra) y validación básica.

5.1. Requisitos Mínimos

1. Crea estructura de carpetas (`data/raw`, `data/processed`, `scripts`, `reports`).
2. Descarga un CSV pequeño de ejemplo (puedes usar `wget` o `curl` a una URL pública, o copiarlo de tu carpeta personal).
3. Genera un reporte en `reports/validacion_inicial.txt` que contenga:
   - Número de filas (`wc -l`).
   - Número de columnas (`head -n1 | tr ',' '\n' | wc -l`).
   - Verificación de valores faltantes (`grep ",,"`).
4. Registra todo en Git con al menos 3 commits significativos.

Criterios de Evaluación

- El script debe ser **idempotente**: ejecutarlo varias veces no rompe nada.
- Usa `mkdir -p` y valida si el archivo ya existe antes de descargar.
- Mensajes de commit claros con prefijos (”feat:”, ”fix:”, ”docs:”).
- Incluye un `README.md` con instrucciones de uso del script.

5.2. Estructura Esperada del Repositorio

```
proyecto_ia/
  data/
      raw/
          datos_sensores.csv
      processed/
  scripts/
      setup_proyecto.sh
  reports/
      validacion_inicial.txt
  .gitignore
  README.md
```

5.3. Plantilla del Script

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

# Crear estructura
mkdir -p data/raw data/processed scripts reports

# Descargar datos (ejemplo)
DATA_URL="https://ejemplo.com/datos.csv"
DATA_FILE="data/raw/datos_sensores.csv"

if [ ! -f "$DATA_FILE" ]; then
  echo "Descargando datos..."
  curl -o "$DATA_FILE" "$DATA_URL"
fi

# Validar datos
echo "=== Reporte de Validación ===" > reports/validacion_inicial.txt
echo "Fecha: $(date)" >> reports/validacion_inicial.txt
echo "Filas: $(wc -l < $DATA_FILE)" >> reports/validacion_inicial.txt
echo "Columnas: $(head -n1 $DATA_FILE | tr ',' '\n' | wc -l)" >> reports/
   validacion_inicial.txt
echo "Valores faltantes: $(grep -c ",," $DATA_FILE || true)" >> reports/
   validacion_inicial.txt

echo "Setup completado. Ver reports/validacion_inicial.txt"
```

5.4. Entrega

- Sube tu repositorio a GitHub (público o con acceso al docente).
- Comparte la URL en el formulario del curso.
- Asegúrate de que el repositorio tenga al menos 3 commits con mensajes claros.

Bonus (opcional): Añade una llamada a Python al final del script para generar estadísticas descriptivas de columnas numéricas y guardarlas en `reports/estadisticas.csv`.

Material de Autoestudio

Fase 4: Estructura de Proyectos Reproducibles

Una buena estructura de proyecto permite que cualquier persona entienda dónde están los datos, el código, los modelos y los reportes.

```
proyecto_ia/
  data/
     raw/           # datos brutos (no limpios)
     processed/     # datos listos para análisis/modelado
  scripts/          # scripts de limpieza, descarga, validación
  notebooks/        # cuadernos exploratorios (Jupyter)
  models/           # modelos entrenados y artefactos
  reports/          # informes, figuras y tablas
  README.md         # descripción del proyecto
```

Fase 5: Integración con Python

Puedes llamar Python desde la terminal para análisis más avanzados:

```
# Ejecutar análisis directo
python - << EOF
import pandas as pd
df = pd.read_csv("data/raw/sensores.csv")
print(df.describe())
print("Temperatura promedio:", df["temperatura"].mean())
EOF
```

Entornos Virtuales de Python

```
# Crear entorno virtual
python -m venv ia_env

# Activar (Linux/macOS)
source ia_env/bin/activate

# Instalar dependencias
pip install pandas numpy matplotlib

# Congelar versiones
pip freeze > requirements.txt

# Desactivar
deactivate
```

Anexo: Alias de Seguridad

```
# Agregar a ~/.bashrc o ~/.zshrc
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

```
alias ll='ls -lh'
alias la='ls -lha'
alias gs='git status'
alias gl='git log --oneline --graph'
```

Siguiente Nivel

En Semana 2 aprenderás vectorización con NumPy y procesamiento eficiente de datasets grandes (>1M filas). Docker y despliegue se cubren en Semana 7-8 (MLOps).