MACHINE LEARNING OPERATIONS

WEEK 4

UNLIMITED

Presented by

Asst. Prof. Dr. Tuchsanai Ploysuwan

Week 4
Undoing Changes

Week 4 Undoing Changes

- Today we're focused on how to undo actions related to git commands and explore historical commits.
- We'll discuss:
  - **git checkout** and **Detached HEAD**
  - **git restore**
  - **git reset**

Week 4
Git Checkout

🎯 สถานการณ์จำลอง

สมมติว่าเราเป็น ML Engineer ที่กำลังพัฒนาโมเดล Classification สำหรับจำแนกดอกไม้ Iris โดยเราได้พัฒนาโมเดลมาเรื่อยๆ ผ่านหลาย versions:

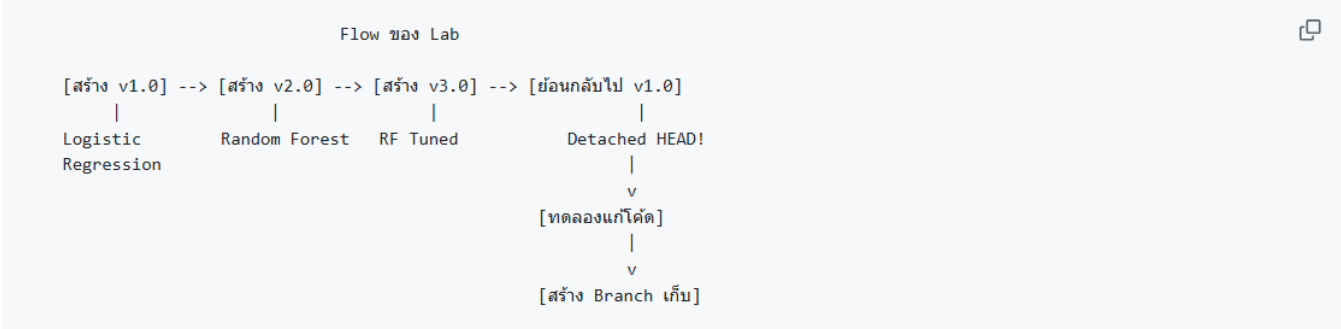
Version	Model	รายละเอียด
v1.0	Logistic Regression	โมเดลพื้นฐาน เริ่มต้นโปรเจค
v2.0	Random Forest	เปลี่ยนอัลกอริทึมเพื่อเพิ่มประสิทธิภาพ
v3.0	Random Forest (Tuned)	ปรับ hyperparameters ให้ดีขึ้น

ปัญหาที่เจอในการทำงานจริง

ในการพัฒนา ML Model มักจะเจอสถานการณ์เหล่านี้:

1. **ต้องการดูโค้ดเก่า** - หัวหน้าถามว่า "โมเดล v1.0 เขียนยังไง?" เราต้องย้อนกลับไปดู
2. **ต้องการเปรียบเทียบ** - อยากรู้ว่า v1.0 กับ v2.0 ต่างกันตรงไหน
3. **ต้องการทดลอง** - อยากลองแก้โค้ดเก่าดูว่าผลลัพธ์จะดีขึ้นไหม
4. **ต้องการกู้โค้ด** - โค้ดใหม่พัง อยากเอาบางส่วนจากเวอร์ชันเก่ากลับมา

สิ่งที่จะได้เรียนรู้ใน Lab นี้



Week 4 Undoing Changes

- **git checkout**
  - This is actually a very versatile command, so versatile in fact, that developers complained it was used for too many different actions, thus new git commands were created, such as **git switch**.

Week 4 Undoing Changes

- **git checkout**
  - A "checkout" is the act of switching between different versions of a target entity.
  - The **git checkout** command can operate on three distinct entities: files, commits, and branches.

Week 4 Undoing Changes

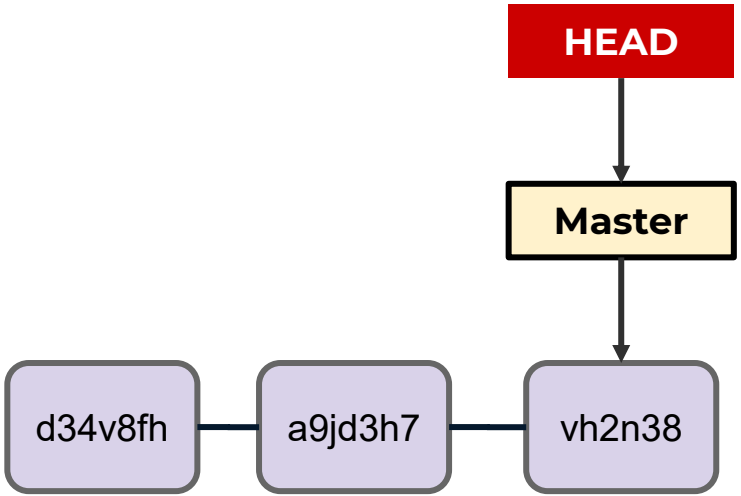
- **git checkout**
  - For example, you could use **git checkout branch_name** instead of **git switch branch_name** to checkout a new branch.
  - Unlike **git switch** however, recall checkout can operate on commits, meaning we can “checkout” historical commits.

Week 4 Undoing Changes

- **git checkout**
  - We can check out a particular commit by specifying its hash, we can get hashes from the **git log** command and we can also see the abbreviated hash using:
    - **git log --oneline**
  - Then we can provide the has as:
    - **git checkout ########**

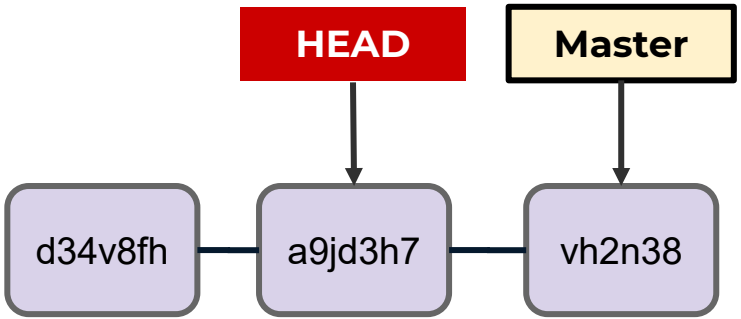
Week 4 Undoing Changes

- Typically our HEAD points to the branch which points to the latest commit.



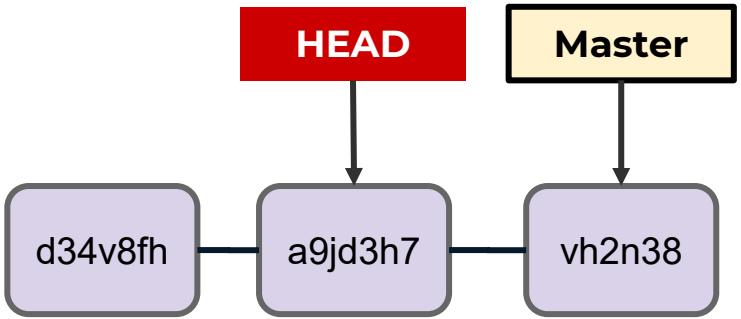
Week 4 Undoing Changes

- Upon calling **git checkout a9jd3h7** we detach the HEAD to a previous commit



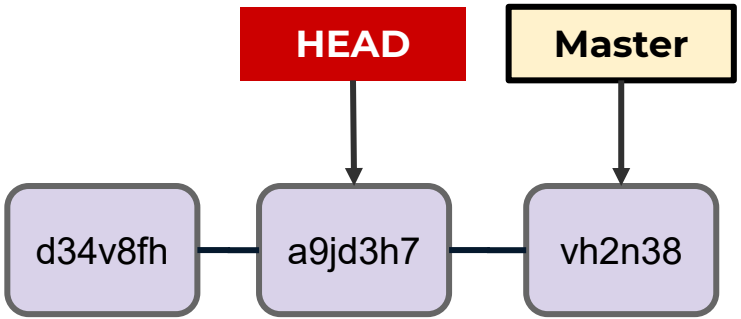
Week 4 Undoing Changes

- You can think of this as traveling back in history to what your code looked like when you ran this commit.



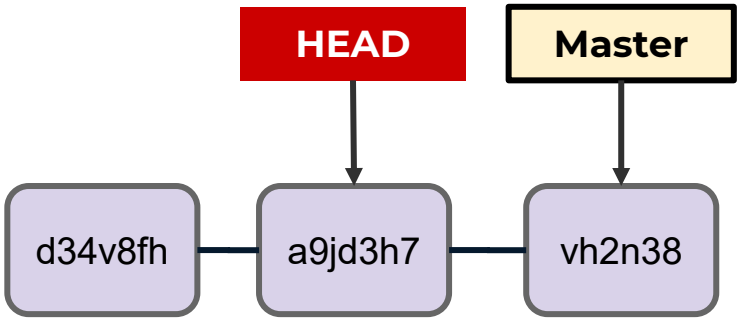
Week 4 Undoing Changes

- This command does **not** undo previous work, you are simply exploring the historical commit.



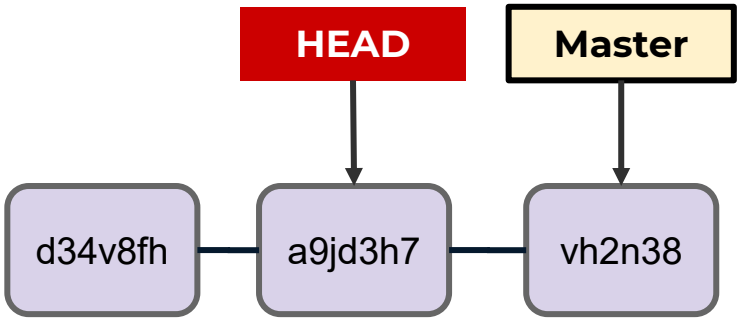
Week 4 Undoing Changes

- If you started making changes here, they won't be preserved since HEAD is not pointing at a branch reference.



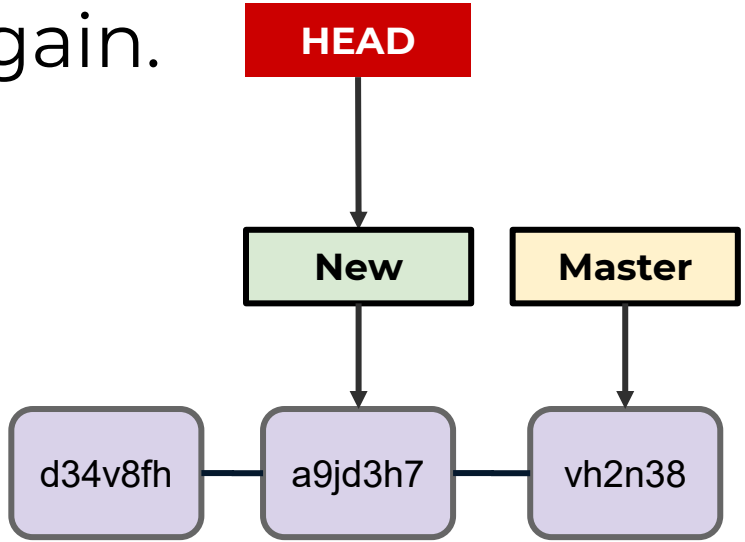
Week 4 Undoing Changes

- However you could create a new branch at this point in time, reattaching HEAD to a branch again.



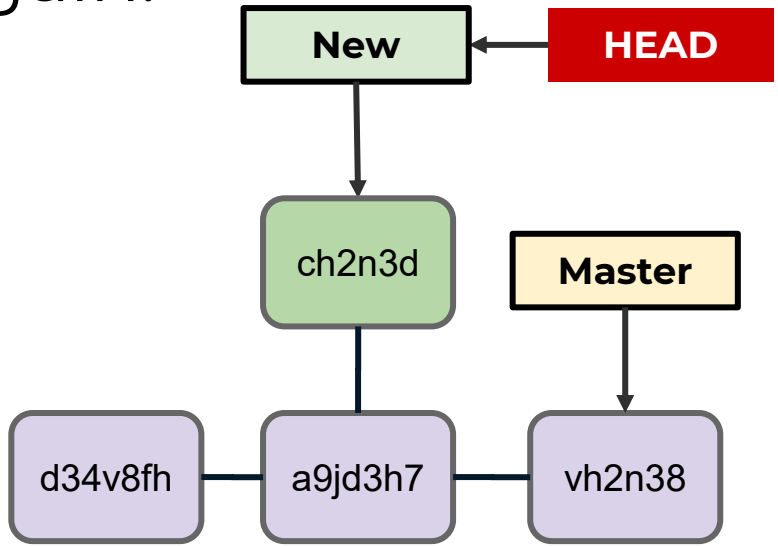
Week 4 Undoing Changes

- However you could create a new branch at this point in time, reattaching HEAD to a branch again.



Week 4 Undoing Changes

- However you could create a new branch at this point in time, reattaching HEAD to a branch again.



Week 4
Git Restore

📖 ทฤษฎี: Git Restore คืออะไร?

🔑 แนวคิดหลัก

ลองนึกภาพว่าคุณกำลังเขียนรายงานด้วย Microsoft Word ถ้าคุณพิมพ์ผิดหรือลบข้อความไปโดยไม่ตั้งใจ คุณจะกด **Ctrl+Z** เพื่อย้อนกลับ ใช่ไหม?

`git restore` ก็ทำหน้าที่คล้ายๆ กัน แต่ทรงพลังกว่ามาก เพราะมันสามารถย้อนกลับไปหาเวอร์ชันใดก็ได้ในประวัติการทำงานของคุณ ไม่ใช่แค่ขั้นตอนก่อนหน้า

🤖 ทำไม Git Restore ถึงสำคัญใน MLOps?

ในการพัฒนา Machine Learning เราทำการทดลองบ่อยมาก และบ่อยครั้งที่ผลลัพธ์ไม่เป็นไปตามที่หวัง:

สถานการณ์	ปัญหาที่เกิด	วิธีแก้ด้วย git restore
ปรับ hyperparameters	Accuracy ลดลง	กู้คืนค่า parameters เดิม
เปลี่ยน preprocessing	Data pipeline พัง	ย้อนกลับไปใช้โค้ดเดิม
ลอง model architecture ใหม่	Training ช้าลง 10 เท่า	กู้คืน model เดิม
แก้ไข evaluation metrics	ผลลัพธ์คำนวณผิด	กลับไปใช้สูตรที่ถูกต้อง

ข้อดีของการใช้ git restore:

- ⚡ กู้คืนได้ทันที ไม่ต้อง copy-paste จากที่อื่น
- 🛡️ ปลอดภัย ไม่มีทางทำให้ประวัติ commit เสียหาย
- 🎯 เลือกกู้คืนเฉพาะไฟล์ที่ต้องการได้

Week 4 Undoing Changes

- **Git restore**
  - We can restore a file to its state at the previous most recent commit using the **git restore command:**
    - **git restore file_name**

Week 4 Undoing Changes

- **Git restore**
  - We can restore a file to its state at the previous most recent commit using the **git restore command:**
    - **git restore file_name**
  - ***Warning:***
    - You can not undo a git restore command, since your changes were not committed!

Week 4 Undoing Changes

- **Git restore**
  - We can restore a file to its state at the previous most recent commit using the **git restore command:**
    - **git restore file_name**
  - ***Warning:***
    - Think of this command as an ultimate “Ctrl+Z” restoring files to their previous commit.

Week 4 Undoing Changes

- **Git restore**
  - Technically speaking **git restore** will restore the file back to the HEAD, which typically we have pointing to the most recent commit in the branch.

Week 4 Undoing Changes

- **Git restore**
  - This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.
  - We state the number of commits from the HEAD to go back to:
    - **git restore --source HEAD~N file.txt**

Week 4 Undoing Changes

- **Git restore**
  - This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.
  - We state the number of commits prior from the HEAD to go back to:
    - **git restore --source HEAD~N file.txt**

Week 4 Undoing Changes

- **Git restore**
  - This actually gives us even more flexibility in our restore procedure, we can restore a file to any commit in the log.
  - We state the number of commits prior from the HEAD to go back to:
    - **git restore --source HEAD~N file.txt**

Week 4 Undoing Changes

- **Git restore**
  - Finally, git restore also allows us to unstage files that we had already added to the staging area using **git add**.
  - We can do this with:
    - **git restore --staged filename**

Week 4 Undoing Changes

- Let’s explore this command in practice:
  - **git restore filename**
  - **git restore --source HEAD~N filename**
  - **git restore --staged filename**

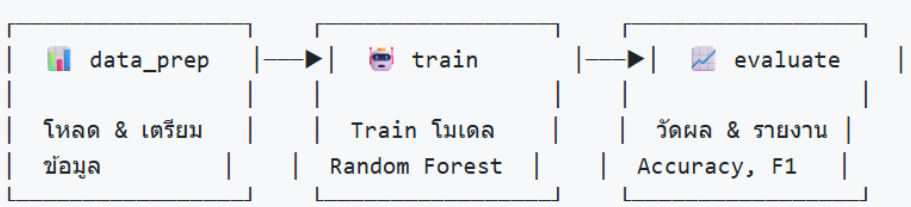
🎬 สถานการณ์จำลอง: เรื่องราวของ Lab นี้

📋 บทบาทของคุณ

คุณคือ **ML Engineer** ที่เพิ่งเข้าทำงานที่บริษัท DataTech ได้ 1 เดือน หัวหน้าทีมมอบหมายให้คุณพัฒนา **Classification Pipeline** สำหรับจำแนกพันธุ์ดอกไอริส (Iris) ซึ่งเป็นโปรเจกต์ฝึกหัดก่อนจะได้ทำโปรเจกต์จริง

🎯 เป้าหมายของโปรเจกต์

สร้าง ML Pipeline ที่ประกอบด้วย 3 ส่วนหลัก:



ไทม์ไลน์ของเหตุการณ์: สัปดาห์แรกของ ML Engineer

วันศุกร์ก่อนหน้า - รับมอบหมายโปรเจกต์

```
┌─ ห้องประชุมทีม ML - 15:00 น. ──────────────────────────┐
│                                                        │
│ 👨‍💼 หัวหน้าทีม: "ยินดีต้อนรับ! เรามีโปรเจกต์ฝึกหัดให้คุณ"  │
│                                                        │
│ 📋 รายละเอียดโปรเจกต์:                                  │
│ ├── ชื่อ: Iris Classification Pipeline                 │
│ ├── Dataset: Iris (150 samples, 3 classes)             │
│ ├── เป้าหมาย: สร้าง Pipeline ครบวงจร                    │
│ └── กำหนดส่ง: วันศุกร์หน้า (7 วัน)                      │
│                                                        │
│ 💭 ความรู้สึก: ตื่นเต้น พร้อมลุย!                       │
└────────────────────────────────────────────────────────┘
```

วันเสาร์-อาทิตย์ - เตรียมตัวและวางแผน

```
┌─ 🏠 บ้าน - วันหยุดสุดสัปดาห์ ──────────────────────────┐
│                                                        │
│ 📚 สิ่งที่ทำ:                                           │
│ ├── ศึกษา requirements และวางแผนการทำงาน               │
│ ├── ออกแบบโครงสร้าง: data_prep.py → train.py → evaluate.py │
│ ├── ศึกษา Iris Dataset                                 │
│ └── ตั้งค่า Git และ Virtual Environment                │
│                                                        │
│ 📝 แผนการทำงานสัปดาห์หน้า:                              │
│ ├── จันทร์: สร้าง Pipeline ทั้ง 3 ไฟล์                  │
│ ├── อังคาร: เพิ่ม Feature Engineering                  │
│ ├── พุธ: ปรับปรุงและ Refactor                          │
│ ├── พฤหัส: ทดสอบและแก้ไข                               │
│ └── ศุกร์: ส่งงาน                                      │
└────────────────────────────────────────────────────────┘
```

วันจันทร์ (Day 1) - สร้าง Pipeline & เกิดปัญหาแรก

```
┌─ 🗓️ วันจันทร์ - สร้าง ML Pipeline ──────────────────────┐
│                                                        │
│ ⏰ 09:00 - เริ่มทำงาน                                   │
│ ├── ☕ ดื่มกาแฟ เปิด VS Code                            │
│ ├── 📁 cd ~/mlops-git-restore-lab                      │
│ └── 📘 เริ่มเขียน data_prep.py                         │
│                                                        │
│ ⏰ 10:30 - Commit #1: Data Preparation ✅                │
│ ├── git add data_prep.py                               │
│ └── git commit -m "feat: add data preparation module"  │
│                                                        │
│ ⏰ 11:00 - พักเบรก 🚶                                   │
│                                                        │
│ ⏰ 11:30 - กลับมาทำงาน                                  │
│ └── 📘 เริ่มเขียน train.py                             │
│                                                        │
│ ⏰ 13:00 - Commit #2: Model Training ✅                  │
│ ├── git add train.py                                   │
│ └── git commit -m "feat: add model training module"    │
│                                                        │
│ ⏰ 13:30 - พักกลางวัน 🍜                                │
│                                                        │
│ ⏰ 14:30 - กลับมาทำงาน                                  │
│ └── 📘 เริ่มเขียน evaluate.py                          │
│                                                        │
│ ⏰ 16:00 - Commit #3: Model Evaluation ✅                │
│ ├── git add evaluate.py                                │
│ └── git commit -m "feat: add model evaluation module"  │
│                                                        │
│ ⏰ 16:30 - ทดสอบ Pipeline                               │
│ ├── 🎉 ทุกอย่างทำงานได้!                               │
│ └── 📊 Accuracy: 96.67%                                │
│                                                        │
│ ═══════════════════════════════════════════════════    │
│ ⏰ 17:00 - 🆘 สถานการณ์ที่ 1 เกิดขึ้น!                  │
│ ═══════════════════════════════════════════════════    │
│                                                        │
│ 💭 "ลองเปลี่ยน hyperparameters ดูดีกว่า"                │
│                                                        │
│ ✏️ แก้ไข train.py:                                      │
│ ├── ตั้งใจจะเปลี่ยน n_estimators=200                   │
│ ├── แต่พิมพ์ผิด: n_estimators=-999                     │
│ └── และ: max_depth="invalid"                           │
│                                                        │
│ ⏰ 17:10 - ลองรันโค้ด                                   │
│ ├── python train.py                                    │
│ └── ❌ ERROR! ValueError: Invalid parameter value      │
│                                                        │
│ 😱 "โค้ดพังหมดแล้ว! ทำไงดี?"                           │
│                                                        │
│ ⏰ 17:15 - 💡 นึกได้! ยังไม่ได้ commit ที่แก้ไข!        │
│ ├── git status → modified: train.py                    │
│ ├── 💡 "ใช้ git restore ได้!"                          │
│ ├── ✅ git restore train.py                            │
│ └── 🎉 โค้ดกลับมาใช้งานได้!                            │
│                                                        │
│ ⏰ 17:30 - เลิกงาน 🏠                                   │
│ └── 📝 บทเรียนวันนี้: git restore ช่วยกู้คืนจากความผิดพลาดได้ │
└────────────────────────────────────────────────────────┘
```

วันอังคาร (Day 2) - Feature Engineering & Add ผิดไฟล์

```
┌─ 🗓️ วันอังคาร - Feature Engineering ────────────────────┐
│                                                        │
│ ⏰ 09:00 - เริ่มทำงาน                                   │
│ ├── ☁️ "วันนี้จะเพิ่ม feature engineering"              │
│ └── 📝 แผน: แก้ไข data_prep.py และ evaluate.py         │
│                                                        │
│ ⏰ 10:00 - แก้ไขไฟล์                                    │
│ ├── 🖍️ data_prep.py: เพิ่ม add_features() function     │
│ │   ├── sepal_ratio = sepal_length / sepal_width       │
│ │   ├── petal_ratio = petal_length / petal_width       │
│ │   ├── sepal_area = sepal_length * sepal_width        │
│ │   └── petal_area = petal_length * petal_width        │
│ │                                                      │
│ └── 🖍️ evaluate.py: เพิ่ม experimental features        │
│     ├── timestamp tracking                             │
│     ├── save_metrics_to_file()                         │
│     └── ⚠️ ยังไม่ได้ทดสอบ!                             │
│                                                        │
│ ═══════════════════════════════════════════════════    │
│ ⏰ 11:00 - 🆘 สถานการณ์ที่ 2 เกิดขึ้น!                  │
│ ═══════════════════════════════════════════════════    │
│                                                        │
│ 🤖 รีบทำ ใช้ git add . โดยไม่ได้คิด                     │
│                                                        │
│ 📋 git status:                                         │
│ ├── Changes to be committed:                           │
│ │   ├── modified: data_prep.py  ✅ พร้อม               │
│ │   └── modified: evaluate.py   ⚠️ ยังไม่พร้อม!        │
│                                                        │
│ 😀 "เดี๋ยว! evaluate.py มี experimental code ยังไม่ได้ทดสอบ!" │
│ ☁️ "ถ้า commit ไปอาจทำให้ production พัง"               │
│                                                        │
│ ⏰ 11:15 - 💡 แก้ไข!                                    │
│ ├── git restore --staged evaluate.py                   │
│ └── 📋 git status:                                     │
│     ├── Changes to be committed:                       │
│     │   └── modified: data_prep.py  ✅                 │
│     └── Changes not staged:                            │
│         └── modified: evaluate.py  (กลับมา working directory) │
│                                                        │
│ ⏰ 11:30 - Commit เฉพาะที่พร้อม                         │
│ ├── git commit -m "feat: add feature engineering to data_prep" │
│ └── 🎉 Commit สำเร็จ!                                  │
│                                                        │
│ ⏰ 11:45 - ตัดสินใจเรื่อง evaluate.py                   │
│ ├── ☁️ "ยังไม่พร้อม ขอกู้คืนเวอร์ชันเดิมก่อน"          │
│ ├── git restore evaluate.py                            │
│ └── 📋 git status → clean                              │
│                                                        │
│ ⏰ 12:00-17:00 - ทำงานต่อ 🍔                            │
│ └── 📝 บทเรียน: ตรวจสอบก่อน add และใช้ --staged เพื่อ unstage │
└────────────────────────────────────────────────────────┘
```

วันพุธ (Day 3) - Over-Engineering & ต้องกลับเวอร์ชันเก่า

```
┌─────────────────────────────────────────────────────────┐
│ 🗓️ วันพุธ - Refactoring                                 │
└─────────────────────────────────────────────────────────┘

⏰ 09:00 - ประชุมกับหัวหน้าทีม
├── 👨‍💼 หัวหน้า: "ความคืบหน้าเป็นอย่างไรบ้าง?"
├── 📊 แสดง Pipeline: data_prep → train → evaluate
├── 👍 หัวหน้าพอใจ
└── 📝 หัวหน้า: "ลองเพิ่ม logging หน่อยนะ"

⏰ 10:00-12:00 - Over-engineering เริ่มต้น! 🔧
├── 💭 "ขั้นทำให้เต็มที่เลย!"
├── 🖍️ evaluate.py:
│   ├── เพิ่ม logging module (DEBUG, INFO, WARNING, ERROR)
│   ├── สร้าง ModelEvaluator class
│   ├── เพิ่ม metrics_history list
│   ├── เพิ่ม file handlers
│   └── เพิ่ม JSON export
│
├── git add evaluate.py
└── git commit -m "refactor: over-engineer evaluation module"

⏰ 13:00 - พักกลางวัน 🍜
└── 💭 คิดว่าทำได้ดีมาก

═══════════════════════════════════════════
⏰ 14:00 - 🆘 สถานการณ์ที่ 3 เกิดขึ้น!
═══════════════════════════════════════════

👨‍💼 หัวหน้า Review โค้ด:
├── 😐 "ทำไมซับซ้อนขนาดนี้?"
├── 📝 "Class ไม่จำเป็น ขอแค่ functions ธรรมดา"
├── 📝 "Logging ก็เกินไป ขอแค่ print ก็พอ"
└── 📝 "กลับไปใช้เวอร์ชันเรียบง่ายเถอะ"

😰 "แต่... commit ไปแล้วนี่!"

⏰ 14:30 - 💡 แก้ไขด้วย git restore --source
├── git log --oneline
│   ├── pqr4567 refactor: over-engineer evaluation module  ← ปัจจุบัน
│   ├── xyz7890 feat: add feature engineering to data_prep
│   ├── abc1234 feat: add model evaluation module  ← ต้องการ!
│   ├── def5678 feat: add model training module
│   └── ghi9012 feat: add data preparation module
│
├── git restore --source=abc1234 evaluate.py
└── 🎉 โค้ดกลับมาเป็นเวอร์ชันเรียบง่าย!

⏰ 15:00 - Commit การแก้ไข
├── git add evaluate.py
└── git commit -m "revert: restore simple evaluation module"

⏰ 15:30 - หัวหน้า Review อีกครั้ง
├── 👍 "ดีมาก เรียบง่าย เข้าใจง่าย"
└── 💡 "จำไว้นะ KISS - Keep It Simple, Stupid"

⏰ 17:00 - เลิกงาน 🏠
└── 📝 บทเรียน: Simple is better than complex
```

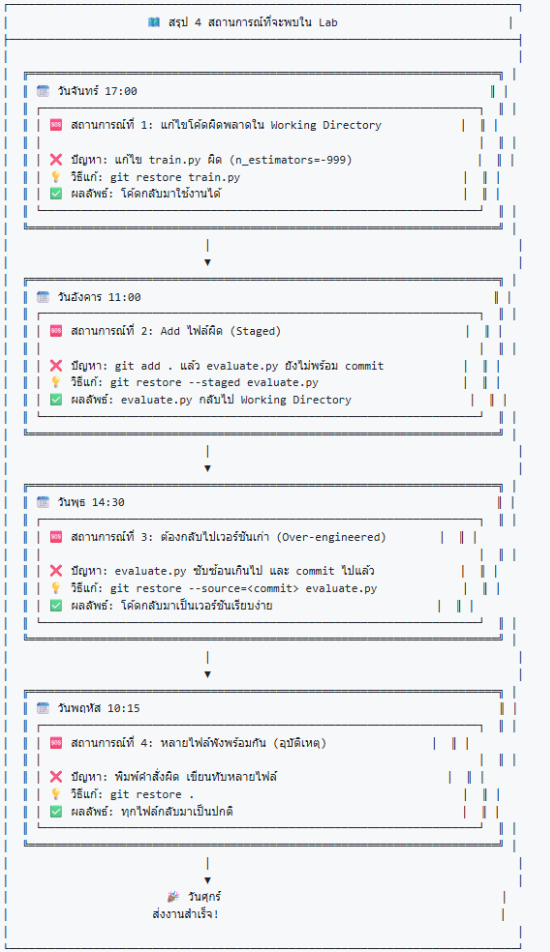
วันพฤหัส (Day 4) - อุบัติเหตุครั้งใหญ่

```
┌─────────────────────────────────────────────────────────┐
│ 🗓️ วันพฤหัส - Testing & Bug Fixing                      │
└─────────────────────────────────────────────────────────┘

⏰ 09:00 - เริ่มทำงานตามปกติ 🤔

⏰ 09:30 - ทดลอง script ใหม่
├── 💭 "ลองเขียน script สำหรับ cleanup"
└── 🖍️ เริ่มเขียน cleanup.sh

⏰ 10:00 - ประชุมด่วน! 📞
├── หัวหน้าเรียกประชุม
├── ⏰ ประชุมเริ่ม 10:30
└── 😰 ริบ! ต้องทำให้เสร็จก่อน

═══════════════════════════════════════════
⏰ 10:15 - 🆘 สถานการณ์ที่ 4 เกิดขึ้น! 🚨
═══════════════════════════════════════════

✨ ริบเก็นไป พิมพ์คำสั่งผิด!

⌨️ ตั้งใจพิมพ์:
   echo "# Clean script" > cleanup.sh

⌨️ พิมพ์จริง (ผิด!):
   echo "# BROKEN!" > data_prep.py
   echo "# BROKEN!" > train.py

😱 ไฟล์หลักถูกเขียนทับ!

📋 git status:
├── modified: data_prep.py
└── modified: train.py

📄 cat data_prep.py:
   # BROKEN!

📄 cat train.py:
   # BROKEN!

😰 "โค้ดหลายร้อยบรรทัดหายหมด! ทำไงดี?!"

⏰ 10:20 - 💡 สงบสติ นึกได้!
├── 💭 "ยังไม่ได้ commit!"
├── 💭 "ใช่ git restore . ได้!"
├── ✅ git restore .
└── 🎉 ทุกไฟล์กลับมา!

⏰ 10:25 - ตรวจสอบ
├── git status → clean
├── python train.py → สำเร็จ!
└── 👍 ทุกอย่างปกติ

⏰ 10:30 - เข้าประชุม (ทันพอดี!) 📞
└── 😊 ไม่มีใครรู้เรื่องอุบัติเหตุ

⏰ 12:00-17:00 - ทำงานต่อ
└── 📝 บทเรียน: git restore . คือเพื่อนที่ดีที่สุดยามคับขัน
```

วันศุกร์ (Day 5) - ส่งงานสำเร็จ 🎉

```
┌─────────────────────────────────────────────────────────┐
│ 🗓️ วันศุกร์ - Delivery Day!                             │
└─────────────────────────────────────────────────────────┘

⏰ 09:00 - ตรวจสอบทุกอย่างครั้งสุดท้าย
├── 🖍️ python data_prep.py → ✅
├── 🖍️ python train.py → ✅
├── 🖍️ python evaluate.py → ✅
└── 📊 Accuracy: 96.67%

⏰ 10:00 - นำเสนอผลงาน 🎤
├── 👨‍💼 หัวหน้าและทีม
├── 📊 Demo Pipeline ทั้งหมด
└── 👏 ทุกคนพอใจ!

⏰ 11:00 - Feedback จากหัวหน้า
├── 👍 "ดีมาก! Pipeline ใช้งานได้จริง"
├── 👍 "โค้ดเรียบง่าย เข้าใจง่าย"
├── 👍 "Git workflow เป็นระเบียบ"
└── 🎁 "พร้อมรับโปรเจกต์จริงแล้ว!"

⏰ 12:00 - ฉลองความสำเร็จ 🍕
└── กินพิซซ่ากับทีม

⏰ 17:00 - เลิกงาน
└── 🎉 จบสัปดาห์ด้วยความสำเร็จ!
```

🗺️ สรุปภาพรวม 4 สถานการณ์



Week 4
Git Reset

Week 4 Undoing Changes

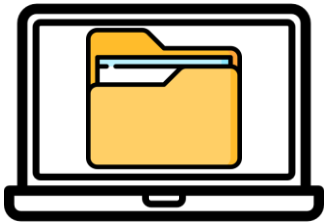
- Git reset allows us to remove commits and “reset” the branch.
- There are two main types of **git reset** calls:
  - **git reset ########**
    - Removes commits in front of the specific hash called, files unchanged.
  - **git reset ######## --hard**
    - Removes commits *and* the changes in the files.

Week 4 Undoing Changes

- To fully understand this, let’s recall our discussions about working directory, staging area, and repository.

Week 4 Undoing Changes

Working Directory



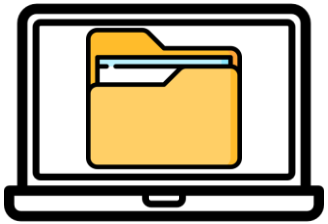
`program.py`

Staging Area

Repository

Week 4 Undoing Changes

Working Directory



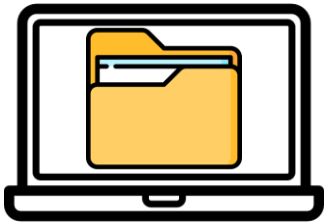
Staging Area

`program.py`

Repository

Week 4 Undoing Changes

Working Directory



Staging Area

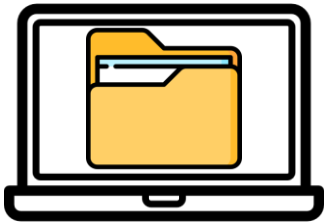
Repository

program.py

“python code”
f3h4782

Week 4 Undoing Changes

Working Directory



index.html

style.css

Staging Area

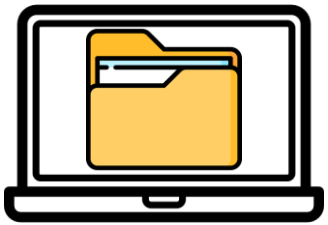
Repository

program.py

“python code”
f3h4782

Week 4 Undoing Changes

Working Directory



Staging Area

index.html

style.css

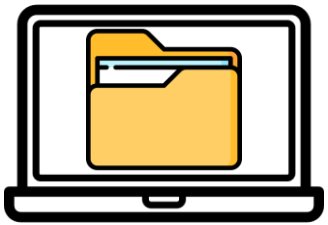
Repository

program.py

“python code”
f3h4782

Week 4 Undoing Changes

Working Directory



Staging Area

Repository

program.py

“python code”
f3h4782

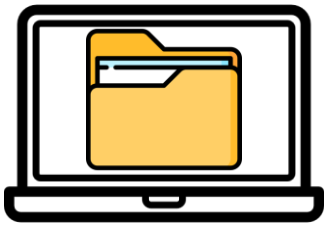
index.html

style.css

“style code”
g82j37l

Week 4 Undoing Changes

Working Directory



Staging Area

Repository

program.py

“python code”
f3h4782

index.html

style.css

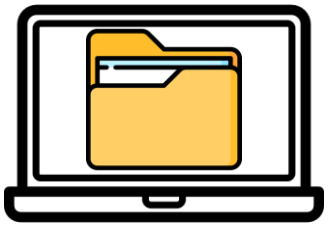
“style code”
g82j371

Week 4 Undoing Changes

Working Directory

Staging Area

Repository



```
>>git reset f3h4782
```

program.py

“python code”
f3h4782

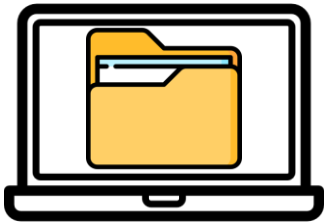
index.html

style.css

“style code”
g82j37l

Week 4 Undoing Changes

Working Directory



index.html

style.css

Staging Area

Repository

program.py

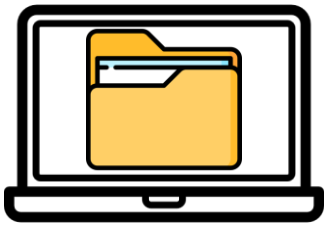
“python code”
f3h4782

Week 4 Undoing Changes

Working Directory

Staging Area

Repository



index.html

style.css

Files are unchanged!
You just reset the commits only.

program.py

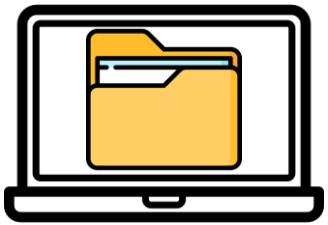
“python code”
f3h4782

Week 4 Undoing Changes

- What if you do want the files to change?
- In the case where you just want to undo everything, including changes and have the branch files look like they did at a previous commit, you add the flag **--hard**.
- For example:
  - **git reset f3h4782 --hard**

Week 4 Undoing Changes

Working Directory



Staging Area

Repository

program.py

“python code”
f3h4782

index.html

style.css

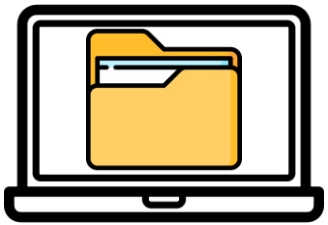
“style code”
g82j37l

Week 4 Undoing Changes

Working Directory

Staging Area

Repository



```
>>git reset f3h4782 --hard
```

program.py

"python code"

f3h4782

index.html

style.css

"style code"

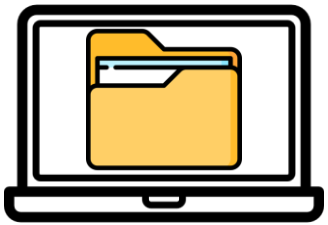
g82j37l

Week 4 Undoing Changes

Working Directory

Staging Area

Repository



program.py

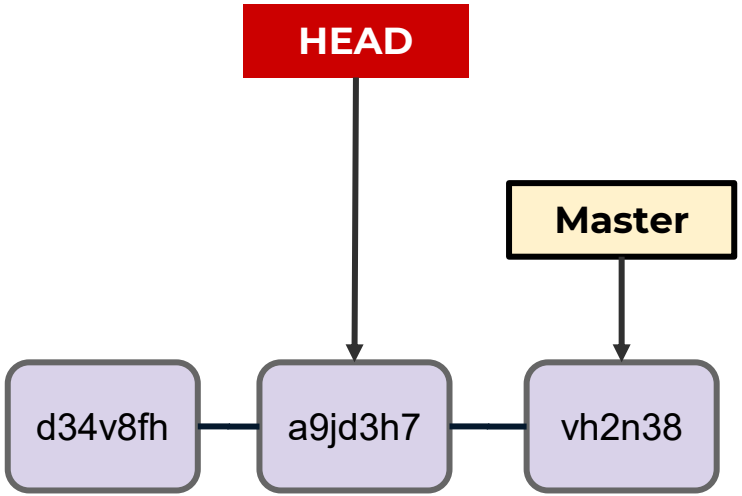
“python code”

f3h4782

```
>>git reset f3h4782 --hard
```

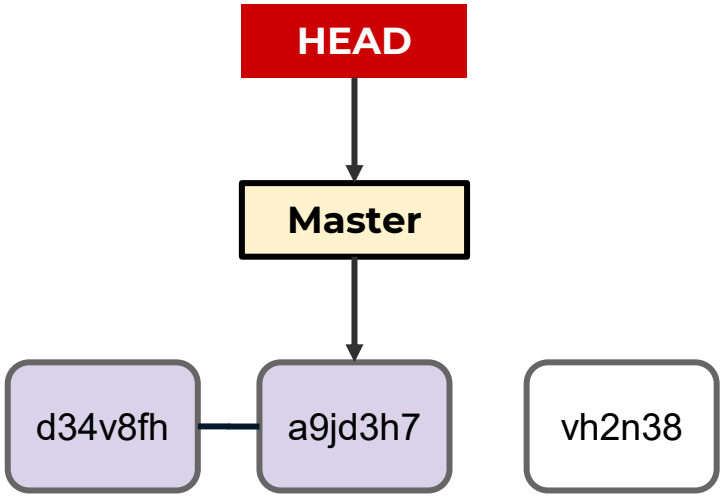
Week 4 Undoing Changes

- We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)



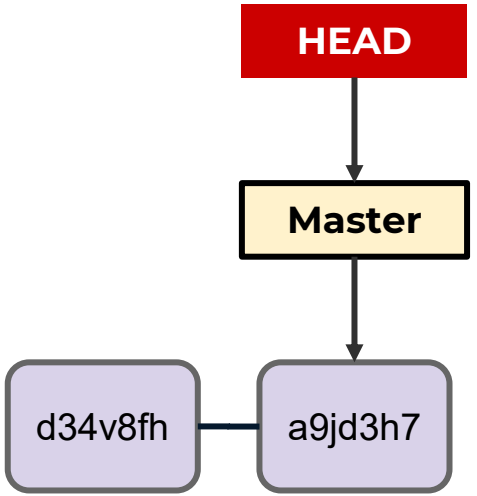
Week 4 Undoing Changes

- We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)



Week 4 Undoing Changes

- We can visualize a **git reset** moving back to a previous commit, but not undoing file changes (unless it is --hard)



Week 4 Undoing Changes

- *Can you undo a git reset --hard?*
  - Technically you can try to recover a commit before Git does its garbage collection, however you should operate under the assumption that a --hard reset is not recoverable.

สถานการณ์จำลองใน MLOps

🎭 บทบาทของคุณในสถานการณ์นี้

คุณเป็น **ML Engineer** ในทีม Data Science ของบริษัทแห่งหนึ่ง ได้รับมอบหมายให้พัฒนา **Classification Model** สำหรับทำนายพฤติกรรมลูกค้า โดยใช้ scikit-learn

🗓️ ไทม์ไลน์ของเหตุการณ์

```
┌─────────────────────────────────────────────────────────────┐
│                    🗓️ TIMELINE OF EVENTS                    │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  วันที่ 1: เริ่มต้นโปรเจค                                   │
│  ─────────────────────────────────────────────────────────  │
│  09:00  ✅ สร้างโปรเจคและ initialize Git                    │
│  10:00  ✅ เขียน config.py กำหนดใช้ RandomForestClassifier  │
│  11:00  ✅ เขียน prepare_data.py สำหรับเตรียมข้อมูล         │
│  14:00  ✅ เขียน train.py สำหรับ train model                │
│  16:00  ✅ เขียน evaluate.py สำหรับประเมินผล                │
│  17:00  ✅ รัน pipeline ครั้งแรก ได้ Accuracy ~92% 🎉       │
│                                                             │
│  วันที่ 2: ปัญหาเกิดขึ้น!                                   │
│  ─────────────────────────────────────────────────────────  │
│  09:00  😤 หัวหน้าบอกว่า RandomForest ช้าเกินไป อยากลอง DecisionTree │
│  10:00  ⚠️ แก้ config.py เปลี่ยนเป็น DecisionTree           │
│  11:00  ⚠️ แก้ train.py ให้ใช้ DecisionTreeClassifier       │
│  12:00  ❌ รัน pipeline ใหม่ ได้ Accuracy แค่ ~75% 😱       │
│  13:00  🤦 หัวหน้าบอกว่าผลแย่ลงมาก ต้องกลับไปใช้ RandomForest! │
│                                                             │
│  🆘 ปัญหา: จะย้อนกลับไปใช้ code เดิมได้อย่างไร?             │
│  💡 คำตอบ: ใช้ Git Reset!                                   │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

🎯 สิ่งที่คุณจะได้เรียนรู้จาก Lab นี้

ขั้นตอน	สิ่งที่จะทำ	สิ่งที่จะได้เรียนรู้
Step 0-4	สร้าง ML Pipeline (Baseline)	การสร้างโปรเจค ML ที่มี version control ที่ดี
Step 5	รัน Pipeline ครั้งแรก	การทดสอบ baseline และบันทึกผลลัพธ์
Step 6	เปลี่ยน Model (ทำผิดพลาด)	จำลองสถานการณ์ที่ทำผิดพลาดในงานจริง
Step 7	พบว่าผลแย่ลง	เข้าใจความสำคัญของการ track changes
วิธีที่ 1	`git reset --hard`	ย้อนกลับและลบทุกอย่างที่ผิดพลาด
วิธีที่ 2	`git reset --soft`	ย้อนกลับแต่เก็บ code ไว้ศึกษา

🔍 ภาพรวมของ Commits ที่จะเกิดขึ้น