APUNTES MAGISTRALES: GIT Y GITHUB

Autor: **Juan Garcia**
Asignatura: Entornos de Desarrollo

Referencia Completa

Abstract

Compendio de referencia que unifica la teoría fundamental de Git (áreas, estados, velocidad) con los comandos prácticos avanzados de GitHub, incluyendo el manejo de ramas, la resolución de conflictos y la navegación en el historial (`reset` , `revert` , `checkout`).

1 Fundamentos: Áreas y Estados de Git

1.1 Las Tres Áreas de un Proyecto Git

Git organiza el trabajo en tres secciones distintas.

- **1. Directorio de Trabajo (*Working Directory*):** La carpeta donde editas los archivos. Los archivos aquí están **Modificados**.
- **2. Área de Staging (*Staging Area* o *Index*):** La zona intermedia donde seleccionas los cambios a incluir en el próximo commit. Los archivos aquí están **Preparados**.
- **3. Repositorio Local (*Git Repository*):** La base de datos oculta (`.git`) donde se guardan todas las versiones confirmadas. Los archivos aquí están **Confirmados**.

1.2 Los Tres Estados de un Archivo

Un archivo pasa por estos estados en el ciclo de vida de un cambio:

1. **Modificado:** El archivo ha sido cambiado, pero Git aún no lo rastrea para la próxima versión.
2. **Preparado (*Staged*):** Marcado con `git add` para ser incluido en el próximo commit.
3. **Confirmado (*Committed*):** Guardado permanentemente en la base de datos local (`.git`).

2 Configuración, Inicio y Flujo de Trabajo Básico

2.1 Configuración de Identidad y Entorno

- **Identidad Global:** `git config --global user.name "Tu Nombre"` | `git config --global user.ema`
- **Verificación:** `git config --list` | `git --version`

2.2 Inicio y Enlace con GitHub

- **Iniciar Repositorio Local (en carpeta vacía):** `git init`
- **Clonar desde Remoto:** `git clone [URL]`
- **Enlazar Remoto (si iniciaste con `init`):** `git remote add origin [URL]`

2.3 Flujo de Trabajo (Aperitivo, Comida y Postre)

1. **Aperitivo (`add`):** Preparar todos los cambios. `git add .` (o `--all`)
2. **Comida (`commit`):** Guardar localmente. `git commit -m "Mensaje"`
3. **Postre (`push`):** Enviar al remoto. `git push`
4. **Revisión:** Usar `git status` entre cada paso.

3 Gestión de Ramas, Versiones y Etiquetas

3.1 Estrategia de Ramas (*Branching*)

La *Memoria* sugiere un modelo de desarrollo robusto con múltiples ramas.

- `main`: Rama de producción, siempre estable.
- `develop`: Rama base para integrar nuevas características.
- `testing`: Rama de pruebas donde se hacen integraciones finales y se marcan versiones.
- **Crear y Cambiar a Rama:** `git checkout -b [nueva_rama]`
- **Moverse a Rama Existente:** `git checkout [rama]`
- **Fusionar Cambios:** Estando en la rama de destino: `git merge [rama_fuente]`

3.2 Etiquetado de Versiones (`tag`)

- **Crear Etiqueta de Anotación (vX.Y):** `git tag -a v1.0 -m "Versión estable con funciones básica`
- **Ver Etiquetas Locales:** `git tag`
- **Enviar Etiquetas a GitHub:** `git push --tags` o `git push origin v1.0`
- **Viajar a una Versión (Temporal):** `git checkout v1.0` (Te pone en un estado *Detached HEAD*).

4 Flujo de Deshacer Cambios y Resolución de Conflictos

4.1 Manejo del Historial y Deshacer

- **Ver Historial Conciso:** `git log --oneline`
- **Volver a Versión Anterior (*Por Archivo*):** Restaura un archivo al commit de la `HEAD` menos N veces. `git checkout HEAD 1 [archivo]`
- **Descartar Cambios Locales (*Working Directory*):** `git checkout -- [archivo]`
- **Revertir un Commit (Seguro):** Crea un commit inverso para anular los cambios, manteniendo el historial. `git revert [hash_commit]`
- **Resetear Historial (Peligroso):** Borra commits posteriores al hash especificado. Úsalo con cautela y solo en local. `git reset --hard [hash_commit]`

4.2 Resolución de Conflictos de Fusión

El conflicto ocurre cuando Git no sabe qué línea mantener porque dos historiales remotos/locales modificaron el mismo lugar (ej., Usuario A y Usuario B suben 'Calculadora.java').

1. **Causa del Conflicto:** Intentar un `push` cuando el remoto tiene cambios que no tienes (`pull` pendiente). Git rechaza el `push` y solicita un `git pull`.

2. **Detección:** El `git pull` falla con un mensaje `CONFLICT (content)`.
3. **Corrección Manual:** Abrir el archivo. Git inserta marcadores de conflicto.
   - `<<<<<<< HEAD` (Tu versión local)
   - `=======` (Separador)
   - `>>>>>>> [hash]` (Versión remota)

   Debes editar el archivo, eliminando los marcadores y dejando solo el código final deseado.
4. **Confirmación de Solución:** Marcar el archivo como resuelto y hacer commit.
   - `git add [archivo_conflictivo]`
   - `git commit -m "Solución de conflicto e integración"`
5. **Sincronización Final:** `git push` para subir el commit de fusión resuelto.

5 Documentación y Exclusiones

5.1 Ficheros de Proyecto

- `.gitignore`**:** Se usa para **excluir** archivos de control de versiones (ej., `*.class`, `*.log`). Las líneas que empiezan con `#` son comentarios.
- `README.md` **(Markdown):** El manual de instrucciones del proyecto. Usa sintaxis simple para encabezados (`#`), negrita ( ), y cursiva ( ).