Evidencia Viernes 23 de Julio (día 05 / Semana 13)

Leonardo Rodenas Escobar

Reflexión:

Como fue un día de presentación de proyectos, decidí avanzar por mi parte, así que realice un mini curso de Git-Github para practicar, pues me sentía débil en ese aspecto. Fue un curso gratificante, y aprendí un par de cosas no vistas en clases que me servirán a futuro.

Link del curso → Click aquí

Apuntes:

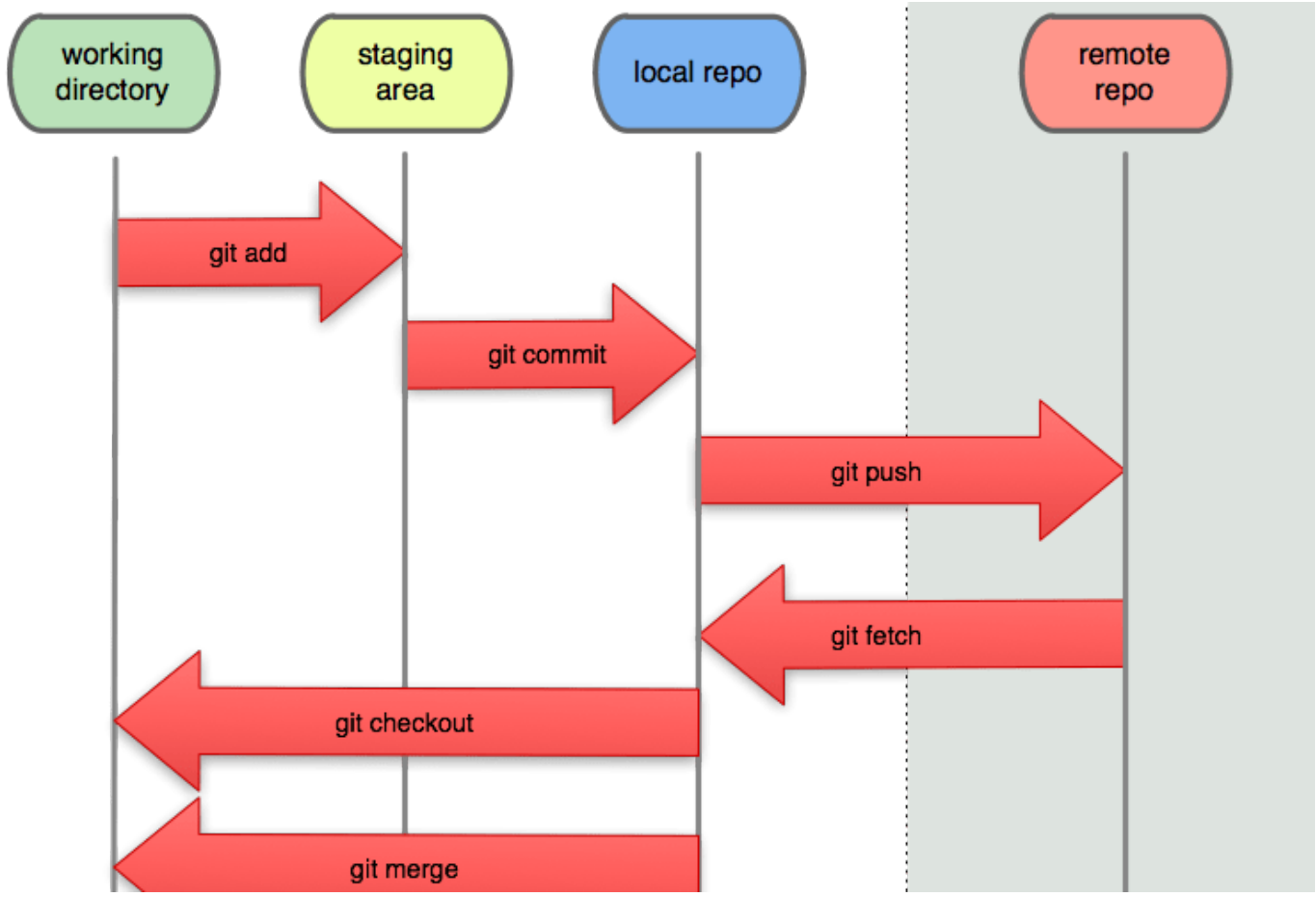
1. Bajado e instalado de Git
2. Creación de carpeta contenedora del repositorio
3. Git Bash dentro de la carpeta
4. Configuración inicial: (se hace solo una vez, después de la instalación)

```
$ git config –- global user.email  (mi correo)
 $ git config –- global user.name  (mi nombre)
```

formas de verificar lo ingresado:

```
$ git config –- global -e
 $ git config –- global -l
 $ git config –- list
```

Áreas de trabajo



5. Crear repositorio local:

- Por consola / ** Se puede hacer manual

** mkdir nombre_carpeta → crea la carpeta en la ubicación asignada
** cd nombre_carpeta → entra en esa carpeta creada
** touch nombre_archivo.extensión → crea el archivo (ej: leo.doc) dentro de la carpeta

```
git init → crea el repositorio local en el pc, se genera una sola vez por proyecto
```

6. Comandos

```
$ git status → estado de seguimiento de archivos (rojo: antes del add . // verde:
esperando el commit)
$ git add . → sube todos los archivos
$git add nombre_archivo → sube solo ese archivo
$ git commit -m “comentario” → se empaquetan los archivos para que puedan quedar
listos y previo a su subida a Github
$ git diff → ve las diferencias de lo que se modifico y lo que esta en el staging
área
```

Si me equivoco al subir algo al stagging area puedo usar esto comando:

```
$ git reset HEAD nombre_archivo
```

De esta manera el archivo del comando de arriba es sacado del staging área y no será subido junto al commit.
Si se generan cambio en el archivo y se quieren deshacer (después de los commit)

```
$ git checkout → borra cambio y vuelve al primer commit
```

7. Visualización del historial de cambiso de Git:

```
$ git log → muestra el commit, el autor y la fecha en que se realizó
```

8. Volver a algo específico dentro de los commit:

- Hacer log
- Anotar el identificador amarillo del commit (letras y numeros = código largo)

```
$ git checkout [código largo] → máquina del tiempo, me devuelve al estado donde
estaba antes de hacer ese commit
```

9. Creación de repositorio remoto:

- Crear cuenta en Github.com
- Ojo: al marcar el readme → creación directo // sin marcar readme → me lleva a las líneas de código de ayuda de Github

```
$git remote add origin htt://sitio_web (extraído del clone repository) → hace la
sincronización remota con la sincronización local.
$ git push -u origin master → se abre la ventana de Github y pedirá
usuario/contraseña (solo una vez, para loguear el pc)
```

10. Crear ramas:

```
$ git branch nombre_rama → crea la rama
$ git branch → me muestra la rama en la que estas trabajando actualmente
$ git checkout nombre_rama → te cambia a la rama donde estabas trabajando
anteriormente
$ git merge nombre_rama → une la rama con la master
$ git branch -d nombre_rama → elimina la rama bifurcada
```

11. Resolver conflictos: Aparecen al trabajar/modificar un archivo en master y en otra rama bifurcada. Cuando se hace el merge sale el conflicto, la razón es que git (en la consola local) no sabe cual de los dos cambios en los archivos dejar. Para solucionar eso, abrir el editor de código y borrar uno de los cambios y luego hacer un nuevo add./status/commit. También se puede ver en la web de Github, ya que en caso de trabajos grupales o colaboraciones, cuando alguien modifique una archivo y luego yo cambie ese mismo archivo, la consola me va a arrojar este mensaje:

```
msg: “Updates were rejected because the remote contains work taht you do    not
have locally”
```

Solución:

Hacer commit, luego git pull origin master (trae al local cambios del colaborador), viendo si hay errores por cambios en la misma linea al hacer el merge, arreglar en editor de código los cambios y luego hacer git add/commit/push

12. Pull: Trae las actualizaciones del Github remoto al local ($ git pull origin master)

13. Pull request (en la web): Se usa para enviar cambios al código del repositorio sin necesidad de que sea miembro del proyecto (que no esté enviado). Se hace entrando en Pull request/Accept Pull request (despues de revisarla, boton verde Github).

14. Ignorar archivos/directorios: para nincluir archivos que no serán tomados en cuenta en el commit. Para esto se crea un archivo .gitignore y se poen dentro lso nombres de los archivos a ignorar. Si es algo dentro de una carpeta poner /carpeta/ para que así ignore todo lo que tenga esa palabra en el nombre.

Así concluye mi avance de hoy, muchas gracias 😄!!!#