**Universidad Mesoamericana de Guatemala**

**Carrera:** Ingeniería en Electrónica, Informática y Ciencias de la Computación

**Materia:** Programación 1

**Catedrático:** Ing. José Arturo Monterroso Muñoz

**Sección:** 1F



**Nombre:** Eduardo Rafael Antillón Jerez

**Número de Carnet:** 202407021

**Fecha de entrega:** 12 / 02 / 2024

**Resumen de los comandos de GitHub**

Comandos de GitHub:

1. Pwd: Imprimir directorio de trabajo
2. Ls: Listar archivos en el directorio
3. Cd: Cambiar directorio
4. ~: Directorio de inicio
5. ..: Subir un directorio
6. -: Directorio de trabajo previo
7. help: Obtener ayuda
8. –h: Obtener ayuda
9. –h: Obtener ayuda
10. man: Manual
11. cat: Salida de los contenidos de un archivo
12. mkdir: Crear nuevo directorio
13. open: Abrir un archivo con el programa asociado, un directorio con un Finder, o una URL con el navegador web por defecto
14. ps: Listar todos los procesos de ejecución
15. kill: Terminar procesos existentes
16. rmd: Archivo permanente eliminado
17. rmdir: Remover directorio

**Trabajando con Git:**

Quick start

1. git clone<url>: Directorio de clonación
2. git checkout –b <new-branch>: Crear nueva rama local
3. git push –u origin <new-branch>: Sincronizar rama local con remota
4. git checkout <branch>: Revisar rama
5. git push origin <branch>: Subir rama a remota
6. git branch –d <branchname>: Elimina rama local
7. git push origin : <branchname>: Elimina rama remota
8. git subtree push - - prefix docs origin gh-pages: Sube documentos como “subtrees” a páginas gh.

**Crear proyecto:**

Proyecto cd/

1. Git init: Inicializa el repositorio
2. Git add. : Añade esos “archivos” desconocidos.
3. Git commit: Nombrar todos los cambios, editar registro de cambios de entrada
4. Git rm - - cached <file>…: Comando ridículamente complicado de deshacer, in caso de que lo olvides. Gitimore

**Derivando y fusionando:**

1. Git branch: Mostrar lista de todas las ramas (\*está activado)
2. Git checkout –b Linux – work: Crear una nueva rama nombrada “linux-work”

<Hacer cambios>

Git commit –a

1. Git checkout master: Volver a la rama principal
2. Git merge Linux-work: Unir conjuntos de cambios desde Linux-work (Git>=1.5)
3. Git pull . Linux-work: Unir conjuntos de cambios desde Linux-work(Todas las versiones Git)
4. Git branch – m<oldname><newname>: Renombrar rama
5. Git branch –m <newname>: renombrar rama actual

**Eliminar Proyecto:**

1. Git branch –d <branchname>: Elimina una rama local
2. Git push origin: <branchname>: Elimina una rama remota
3. Git remote prune <branchname>: Actualiza sincronización local/remota

**Fusionando “Upstream”:**

1. Git remote –v: Obtener lista de ramas remotas
2. Git remote add upstream <upstream github url>: Añadir original como upstream
3. Git remote –v : Revisar Upstream
4. Git fetch upstream: Obtener repositorio original
5. Git checkout development: Cambiar a rama principal en bifurcación local
6. Git merge upstream/development: Fusionar original con bifurcaciones
7. Git diff - - name-only | uniq | xargs subl: Reparar conflictos en Texto Sublime

**Parches importantes:**

1. Git apply < .. /p/foo.patch
2. Git commit –a

**Exportación de parches:**

<make changes>

1. Git commit –a –m “commit message”
2. Git format-patch HEAD^: Crea 0001-commit-message.txt. (HEAD^ significa cada parche desde la revision antes del extremo de la rama, también conocido como HEAD )

**Inspeccionando Revisiones:**

Inspeccionar historial visualmente

1. Gitk: Esto abre una ventana Tk, y muestra cómo las revisiones están conectadas

Inspeccionar historial

1. Git log: Esto canaliza un registro de la rama actual en tu PAGER
2. Git log –p: Del mismo modo, pero adjunta un parche después de cada mensaje de designación.

Inspeccionar una designanión específica

1. Git show HEAD: Muestra información de designación, diffstat y parches del extremo de la rama actual

**Referente a revisiones:**

Por nombre

1. Git log v1.0.0: Muestra historial que conduce a etiqueta “v.1.0.0”
2. Git log master: Muestra historial de rama “principal”

**Relativo al nombre:**

1. Git show master^ : Muestra “padre” a última revisión de “master”.
2. Git show master~2: Muestra “abuelo ” en el extremo de “master”.
3. Git show master ~3: muestra “bisabuelo” en el extremo de “master” (Captas la idea)

Por salida de “git describe”

1. Git show v1.4.4-g730996f: Obtienes esta cadena mediante la llamada “git describe”

Por “hash” (internamente todos los objetos son identificados como “hash”)

1. Git show f665776185ad074b236c00751d666da7d1977dbe
2. Git show f665776: un único prefijo es suficiente

**Etiquetar una revisión:**

1. Git tag v1.0.0: Hacer que el HEAD actual sea conocido como “v1.0.0”.
2. Git tag interesting v1.4.4 – g730996f: Etiqueta una revisión específica (No HEAD).

**Comparando revisiones**

Diferencia entre dos revisiones

1. Git diff origin..master: introduce una diferencia en PAGER
2. Git diff origin..master>my.patch: Introduce una diferencia en my.patch

**Obtener diferencia de un trabajo no nombrado**

1. Git diff - - stat HEAD

**“Sublime” como editor de texto predeterminado**

1. Cd~
2. Mkdir bin
3. Ln –s “/Applications/Sublime Text 2.app/Contents/SharedSupport/bin/subl” ~/bin/subl
4. Git config - - global core.editor “subl –n –w”

**Si eso no funciona**

1. sudo rm –rf /usr/local/bin/subl
2. sudo ln –s /Applications/Sublime\Text\ 2.app/Contents/SharedSupport/bin/subl/usr/local/bin