Conceptos GIT:

Git es un CVS o sistema de control de versiones. Es necesario cuando varios programadores trabajan en los distintos ficheros de un proyecto común.

Git y otros CVS, se pueden usar a menudo a partir de un GUI, pero en el fondo trabajan con comandos, por lo que es importante conocer al menos un CVS por comandos para saber de su potencial. En este caso se explica GIT en comandos.

Un concepto fundamental de un CVS es el de repositorio. Se trata de un área de almacenamiento en la que se encuentran los ficheros de un proyecto, de tal suerte que en Git tenemos un repositorio remoto (en la nube) y un repositorio local en nuestro ordenador que contendrá una copia del contenido del repositorio remoto.

En líneas generales suele haber un repositorio remoto para un proyecto, pero cada miembro del equipo que trabaja en dicho proyecto, tendría su repositorio local, es decir cada programador trabaja con su copia local.

Lógicamente Git ofrece los comandos necesarios para que los cambios efectuados por un programador en el proyecto (cambios en su repositorio local) se puedan transmitir al repositorio remoto, de modo que el resto de miembros del equipo puedan trabajar con la versión más actualizada del proyecto, esto es, Git también permite usar comandos para bajar la versión actual del repositorio remoto a tu repositorio local.

También permite operaciones más avanzadas, como volver a una versión anterior de tu trabajo si fuera necesario. De hecho se llama snapshot a una foto del estado actual de tu trabajo, y será muy común trabajar en Git a través de la creación de estos snapshots.

Cuando se instala Git, lo haremos en la versión consola como se ha dicho, y esta consola de Git se llama bash. Este nombre bash procede del SO Unix, de hecho sus comandos son al estilo de Unix (esto es, por tanto, como en Linux y otros SOs derivados de Unix). Esto implica que aunque instalemos Git en Windows, realmente no usamos el CMD de Windows, sino el bash de Git.

En Git distinguimos 4 áreas para nuestros ficheros de proyecto:

Working Directory:
- Es el directorio local normal en el que trabajamos nuestro proyecto.

Staging Area:
- Es un mapa de trackeo o seguimiento, que incluye los ficheros del working directory que necesitamos que Git procese. Los ficheros que no se incluyan en esta área (luego se verá el comando para incluir ficheros) serán ignorados por Git y se comportarán como si estuvieran en cualquier otra carpeta.

Local Repository:
- Similar a una bandeja de entrada/salida, un área donde se persisten los ficheros trackeados de forma empaquetada en snapshots. Este repositorio local se comunica con el remoto cuando lo necesitemos, para exportar nuestras actualizaciones del proyecto o importar las de los demás miembros del equipo.

Remote Repository:
- El repositorio remoto está en la nube, prestado por algún servicio web, como se ha dicho permite comunicar los cambios locales de un miembro al resto del equipo. En nuestro caso dicho servicio web será GitHub.

Gestiones básicas:

En github.com > Sign Up, te registras para este proyecto (bajo plan gratuito). Crear una cuenta cada miembro de equipo, si un miembro tiene por iniciales ABC y pertenece al Team 8, sería "rawson-team8-abc". Esto permitirá crear informes de actividad adecuados y trabajar todos con la misma configuración de base en la cuenta, además os identificaré y agregaré a los repositorios públicos que os correspondan.

Para bajar Git en sí, vas a git-scm.com, descargas para tu SO, y en la instalación te aseguras de seleccionar el Git Bash. Probablemente lo mejor sea seleccionar "usar Git sólo desde el bash" en lugar de "integrar con el CMD de Windows" en el caso de Windows. Seleccionamos la Open SSL. Los de Windows daremos a check-out Windows style. Seleccionamos MinTTY como emulador de consola para el bash.

Para abrir el bash desde el working directory tenemos que hacer shift+clic derecho en el directorio y Git Bash here.

Una vez en el bash recordar comandos Unix básicos: ls (lista contenido dir actual), pwd (dice el path actual), cd (cambiar de directorio).

Los miembros de cada equipo deben hacer pruebas de Git (asegurarse de que saben usarlo y los miembros se van a entender entre ellos con Git), pueden crear para el equipo su repositorio GitHub público de testeo personal, digamos TestTeam1: Por ejemplo el miembro portavoz del equipo (u otro miembro) puede crear el repositorio público citado, esto es... dar al +, New repository, das nombre, descripción, añades readme, dejas el ignore, y seleccionas público como modalidad. Le das una licencia por ejemplo una GNU y listo. Esto te lleva a una web con una URL que es la remote URL que permitirá a Git enlazar tu working directory con el repositorio remoto. Luego pueden hacer pruebas clonando el repositorio público al local de cada miembro, y hacer pruebas de commits, etc.

Comandos GIT (resumen: completar y practicar con otras fuentes* *ver abajo*):

git init	*Indica a Git que vamos a empezar a usarlo en un proyecto dado, si ejecutamos este comando en un directorio llamado a ser working directory, creará en este un fichero .git que no tenemos que tocar.*
git config --global	*Llamar con user.email "email" y luego con user.name "nombre", para configurar el desarrollador que trabaja desde ese equipo.*
git add	*Si le pasamos el nombre directo de un fichero, lo añade al staging área y comienza a ser trackeado, si hacemos "git add ." añade al staging todos los ficheros del WD. Sin embargo en cuanto cambiemos un fichero requiere sufrir un add de nuevo si no queremos que status nos lo muestre como modificado, y además para comitear los cambios en caso positivo. Añadir ficheros al staging area se dice también indexarlos o añadirlos al INDEX tal cual, y es como un pequeño "guardar partida" para ese fichero, como un mini-commit, lo que ocurre es que un commit embebe todos los add en un paquete y es una foto no de un fichero concreto sino del estado del proyecto, y a parte el commit lo lleva al repo local que es una lanzadera para el repo remoto.*
git diff	*Nos dice los cambios concretos producidos en el working directory desde su última adición al staging área. Se usa en conjunción con checkout.*
git status	*Nos da el estado de los ficheros del proyecto, si están en un área u otra, etc.*
git commit	*Con <git commit -m "mensaje"> pasamos del staging al repo local. Crea snapshots.*
git checkout -- nombre_fichero	*Provoca revertir los cambios últimos hechos sobre el fichero, entendemos últimos cambios aquellos que se produjeron desde el último add, es decir checkout es para cuando te arrepientes de los ultimos cambios a un*

fichero, por tanto el add se debe usar como un "guardar partida" básico. Si en el tiempo de edición errónea de un fichero, también creaste nuevos ficheros, revertir el fichero erróneo sólo modifica el erróneo, es decir no implica borrar los ficheros acompañantes que creaste en esa sesión.

git log *Te muestra el listado de commits realizados (lista de snapshots), de la rama en la que estemos, pues cada rama tiene su lista, aunque puede pasar que una rama en su log muestre sublista de snapshots que heredó de la rama desde la que se creó la nueva rama, cada snapshot tiene su código HASH identificativo, su autor, la fecha, su mensaje, donde está el HEAD o cabezal de navegación por snapshots, es decir qué versión del proyecto tenemos en el working directory.*

git remote add origin URL (URL sin comillas y debe acabar en .git).

Se le pasa la URL de GitHub que apunta a un repositorio remoto. La idea es comenzar a participar en el repositorio remoto a partir de lo que nosotros ya tenemos en local, habitualmente clonas el remoto en local y a patir del estado de proyecto que recibes continúas el trabajo.

git push *Para mandar del local al remoto, te pide si lo usas así tal cual, email y contraseña para identificarte como colaborador.*

git push origin master

Teniendo GitHub abierto, abrirá un formulario web de login para poder habilitar a tu bash a subir cambios del repo local al remoto. Tienes que meter tu login/passwd de GitHub. "master" podría ser otra rama si no es la master (la main que consolida el proyecto definitivo), y con este push se trata de subir al remoto el estado de esta rama en local, o incluso subir la rama en sí completa si no la habíamos subido al repo remoto aún.

git pull *Baja y fusiona los cambios remotos del repo de la nube al local.*

git clone *Vamos a GitHub, el repositorio creado buscamos el botón verde de Clone or Download, copiamos la URL, creamos en nuestro equipo de trabajo un directorio de importación, abrimos el bash de Git en esa ubicación, y escribimos git clone URL_ANTERIOR, de este modo podemos empezar a trabajar en ese repositorio. Lo primero será ir en el bash a la subcarpeta creada que será el working directory real. El clone se hace para empezar a trabajar en un proyecto que ya se ubica en un repositorio remoto.*

git fetch origin (ENTER, y después...) git reset --hard origin/master

Renunciaría a todos los cambios producidos en tu repo local, no sólo en staging área sino los commits incluso, y recuperaría la versión del proyecto del repo remoto.

git merge *Si estás en la rama X, y haces git merge Y, Git intenta fusionar la rama Y con la X. Se usa cuando una rama alternativa se confirma como rama evolutiva del proyecto válida y quieres por ejemplo fusionarla con la master. Sin embargo este proceso tanto con merge como con pull puede producir conflictos, en cuyo caso hay que fusionar manualmente dichos ficheros que mostrará Git, que por tanto derivará en producir ficheros modificados según status, y por tanto hay que hacerles add de nuevo.*

git diff <source_branch> <target_branch>

Si haces un diff sobre ramas, te ayudará a saber las diferencias.

git branch *Muestra las ramas que hay y en cuál estamos ahora, cada rama es una línea independiente del proyecto, que al momento de su creación hereda todo el historial de snapshots de la rama desde la que se crea la nueva rama.*

git branch nRama *Crea la nueva rama nRama heredando lo de la actual.*

git ckeckout rama *Cambia a la rama elegida, provocando que el working directory pase a mostrar lo que tenía esa rama (sin perder nada de la rama desde la que saltamos).*

git checkout -b feature_x

Es lo más cómodo porque crea la rama feature_x a partir de la actual, y automáticamente salta a ella.

git branch -d feature_x
Borra la rama feature_x.
git tag tag_token commit_id
Taguea el commit, para que en git log aparezcan tags que añadan alguna info adicional al mensaje del commit.

GIT Ignore:

Se puede crear en el working directory un fichero .gitignore de texto, donde ponemos la ruta relativa de los directorios y ficheros que queremos que Git ignore, esto es útil para por ejemplo no tener que estar sincronizando ni subiendo ni bajando del repo remoto ficheros como el directorio build de un proyecto NetBeans, por lo que convendría crear este fichero .gitignore y marcar ese directorio build, aquellos miembros cuyas entregas impliquen usar NetBeans. El propio .gitignore es un fichero que sufrirá el proceso como los demás, es decir add y demás.

Otras fuentes:

Guía sencilla Git -> http://rogerdudler.github.io/git-guide/index.es.html
Comandos de Git en vídeo -> https://www.youtube.com/watch?v=QGKTdL7GG24