# Cheatsheet de comandos Git

## Comandos básicos

**Inicializar repositorio**
Para inicializar un repositorio o primeiro é crear un directorio/carpeta co nome que queiramos darlle e despois entrar no directorio. A continuación:

```
git init
```

Isto creará unha carpeta oculta .git con toda a información de xestión do repositorio

**Configurar repo**
Consulta a configuración de git:

```
git config -l
```

O básico é configurar nome de usuario e email. Pode facerse de modo global, para todos os repos do teu sistema, ou de xeito local, para o repo actual:
Configuración global

```
git config --global user.name "o_meu_nome"
git config --global user.email "o_meu_correo@o_meu_dominio.com"
```

Configuración local

```
git config user.name "o_meu_nome"
git config user.email "o_meu_correo@o_meu_dominio.com"
```

Recorda: a configuración local só afecta ao un repo, a configuración global a todos os do sistema (excepto aqueles para os que definises configuración local ;-)

**Consultar estado**
En calquera momento podemos consultar o estado do noso repositorio e saber se hai cambios etc.

```
git status
```

**Trackear ficheiros – "Seguirlles a pista"**
Despois de crear un novo ficheiro ou realizar algunha modificación sobre ficheiros dentro do noso repositorio "git" darase conta de que houbo cambios (podemos comprobalo con git status).
Se nos interesa "trackear" eses cambios, de cara a engadilos a unha futura "confirmación" ou commit entón temos que engadilos á área de "stage":

```
git add nome_ficheiro
```

é moi común engadir varios ficheiros engadindo varios nomes ou mesmo todos os ficheiros modificados do directorio actual de traballo con:

```
git add .
```

**Commit / Confirmación de cambios**
Un "commit" é como un punto de confirmación, un punto no tempo onde se conxelan o estado actual do noso código atendendo aos cambios indicados na "staging área" que indicamos mediante "git add".

```
git commit
```

Se queremos indicar directamente a mensaxe de comentario e evitar que nolo pida un programa externos utilizamos o seguinte código:

```
git commit -m "mensaxe de resumo do commit"
```

**Log**
Git garda un histórico de cambios, prestando especialmente aos commits ou puntos de confirmación. Podemos consultar o histórico destes puntos con git log:
```
git log
```
Algunhas opcións interesantes son as seguintes:
```
git commit --oneline
```
Opción moito menos “verbose” que permite ver históricos grandes dun xeito resumido.
```
git commit --all
```
Podemos ver todo o histórico, incluso desde o pasado. ¿?

**Checkout (viaxar no tempo)**
Con esta opción podemos movernos entre os diferentes commits.
Como primeiro paso podes facer un "git log –oneline" para ver a lista de commits e a continuación o checkout:
```
git checkout identificador_de_commit
```

# Xestión de branches/(ramas/polas)

**Mostrar branches**
Mostra unha lista das branches, resaltando na que estamos situados.
```
git branch
```

**Crear branch**
Podemos crear unha nova branch indicando o nome
```
git branch nova_branch
```
Cando creamos unha nova branch non nos situamos nela, para iso podemos utilizar o comando checkout coa opción -b (de branch):
```
git checkout -b outra_branch
```

**Mudar de branch**
Para movernos dunha branch a outra utilizamos o comando checkout:
```
git checkout nome_branch
```

**Fusión/Merge de branches**
Utilizamos as branches para realizar desenvolvementos de xeito paralelo e, polo xeral, remataremos fusionando as novas branches con master/main, ou con outra branch. O xeito de facer isto é situarse na branch "principal" e desde aí facer un "merge" á outra branch:
```
git checkout branch_principal
git merge branch_que_queremos_mergear
```

**Merge Fast-Forward vs Three-way**
En función da situación das branches a fusionar poden darse dous casos: merge fast-forward / FF / avance rápido ou Three-way / 3-way / recursivo

Nesta explicación partimos de que estamos a traballar con dúas branches: main e nova_branch

O merge Fast-Forward dase cando facemos merge de dúas branches das cales unha parte directamente da outra, é dicir, a nova branch engadiu commits mais a branch orixinal non se modificou.
Este caso de merge non implica novos commits, unicamente o punteido de HEAD → *main* avanza ata o punteiro de *nova_branch*

Exemplo de código (merge Fast-Forward)

```
(main) > checkout -b nova_branch
(nova_branch) > touch file1.txt
(nova_branch) > git add .
(nova_branch) > git commit -m "file.txt added"
(nova_branch) > switch main
(main) > git merge nova_branch
```

O merge Three-Way dase cando, ao contrario que no caso de FF, houbo cambios en ambas as ramas que se queren fusionar. Neste caso, facer un merge suporá a creación dun novo commit que fusione os cambios feitos por ambas branches. Esta fusión pode dar lugar a conflitos que, dado o caso, haberá que solucionar de xeito manual, seguidos dun novo commit.

Exemplo de código (merge Three-way)

```
(main) > checkout -b nova_branch
(nova_branch) > touch file1.txt
(nova_branch) > git add .
(nova_branch) > git commit -m "file.txt added"
(nova_branch) > switch main
(main) > touch file2.txt
(main) > git add .
(main) > git commit -m
(main) > git merge nova_branch
```

Exemplo de código (merge Three-way)

```
(main) > checkout -b nova_branch
(nova_branch) > echo "pan" > lista_compra.txt
(nova_branch) > git add .
(nova_branch) > git commit -m "pan added"
(nova_branch) > switch main
(main) > echo "queixo" > lista_compra.txt
(main) > git add .
(main) > git commit -m
(main) > git merge nova_branch

>>> Neste caso teremos un conflito, porque nas dúas branches se escribe sobre o
mesmo ficheiro. Haberá que editar o ficheiro en conflito e facer un novo commit.
>> nano lista_compra.txt

(main) > git add lista_compra.txt
(main) > git commit -m "lista da compra completa"
```

# Colabora nun repo

Un dos xeitos de colaborar en proxectos é aportando o teu código. Ben sexa por estar traballando no mesmo proxecto como por querer facer unha achega a un repositorio git antes ou despois terás que enviar o teu código para fusionalo co código orixinal.

Veremos un exemplo de proceso típico de traballo contra un repositorio Github.

**Fork do repositorio orixinal**
Desde a túa conta de github realizas un "fork" do repositorio principal, polo que na túa conta pasas a ter un repositorio copia do orixinal.

**Clonación a local**
Clona a túa copia do repositorio orixinal
```
git clone enderezo
```

**Edición de código**
Crea unha nova branch para editar o código, crear novos ficheiros, etc e sitúate nela
```
git checkout -b nova_branch
```
(ou)
```
git branch nova_branch
git checkout nova_branch
```

Edita o código
Realiza un commit
```
git add .
git commit -m "os meus cambios"
```

**Actualizar nova branch**
Actualiza a nova branch contra a túa copia do repositorio
```
git push origin nova_branch
```

**Crear Pull-Request**
Diríxete á túa copia do repositorio en GitHub e solitica Pull Request
- Compare & Pull Request
- Engade un comentario
- Crea Pull-Request

**Espera de confirmación**
Terás que esperar a que a persoa propietaria do repositorio orixinal acepte (ou rexeite) a túa proposta de cambios.

**Actualización da túa copia do repositorio**
Se aceptan (e 'mergean') os teus cambios propostos a túa copia do repositorio non estará aínda actualizada.
Realiza, a través da web "Fetch and merge upstreams" para actualizar a túa copia

**Actualización do repositorio local**
Finalmente actualiza do teu repositorio local (vs a túa copia)
```
git pull
```