# Summary

## Onedrive vs git

|  |  |
| --- | --- |
| **Anydrive service** | **Git** |
|  |  |

Shape, arrow

Description automatically generated

## Checking global options

**git config --list:** Shows what the settings.

* git config --system --list: settings for every user on this computer.
* git config --global --list: settings for every one of your projects.
* git config --local --list: settings for one specific project.

git config --list --show-origin: Display all the global options we have.

git config user.name: Shows the username of the computer.

git help <verb> or man gitignore

## Setting global options

git config --global user.name <”username”>: Changes our username.

git config --global user.email <user@email.com>: Changes our user email.

git config --global core.editor <notepad or text\_editor>: Changes our text editor for commits.

git config --global init.defaultBranch main: Changes the name of the production branch.

### Listing files we don’t need to track

**.gitignore:** we can list in this files all patterns we don’t want to commit.

https://github.com/github/gitignore/blob/main/R.gitignore

| **Symbol** | **Meaning** | **Symbol** | **Meaning** |
| --- | --- | --- | --- |
| # | To ignore a line | \* | Zero of more characters |
| /pattern | To avoid recursivity | ? | Matches a single character |
| pattern/ | to specify a directory | [1-9] | Matches any character between them |
| !pattern | Negate a pattern | \*\* | Nested directories  a/\*\*/z -> a/z, a/b/z, a/b/c/z |

## Starting to work

|  |  |
| --- | --- |
| **Starting a new project** | **Continue a project** |
| 1. Create a new folder to use it in the project and set it as working directory.  mkdir /documents/directory\_name  cd /documents/directory\_name | 1. Set the parent folder as working directory  cd /documents |
| 2. Say to git that you want to track that folder  **git init [-b <branch-name>] [<directory>]** | 2. Get the master branch of the remote in a folder  git clone [-o <remote\_name>] [-b <branch\_name>] <remote\_url> [< directory\_name>] |

## Committing changes

### Fill staging area.

git add <files-names or patterns>

### Check file’s modification status

git status -s

| **Staging area** | **Working tree** | **Effect** |
| --- | --- | --- |
| ? | ? | New files that aren’t tracked |
| A |  | New files that have been added to the staging area |
|  | M | modified in the working directory but not yet staged |
| M |  | Modified and staged |
| M | M | Modified, staged, and then modified again |

### Check changes

|  |  |
| --- | --- |
| Text  Description automatically generated  From line 58, 13 lines have been removed  From line 58, 6 lines have been added | From 1, 1 line have been modified |

|  |  |
| --- | --- |
| **Staging area** | **Actions** |
| git diff <hash1>..<hash2> | Sees the changes between two commits. |
| git diff <path> | Gets the difference between ***a specific file or folder*** modified in working directory and staging area. |
| git diff | Gets the difference between ***all files*** modified in working directory and staging area. |
| git diff -r HEAD [path] | To compare last commit with in the staging area  -r flag means "compare to a particular revision". |
| git diff --cached <path>  git diff --staged <path> | Gets the difference between ***a specific file or folder*** modified in staging area and commits. |
| git diff <hash\_old> <hash\_new> | Gets the difference between ***all files*** modified from one commit to other. |
| git difftool | It is a frontend to git diff and accepts the same options and arguments |

### Commit changes

**git commit [-a] [-v] [-m <’comment’>]:** By default, opens a notepad to write text describing the commit then takes all the changes in the staging area.

* **[-a]:** Commits all tracked files no matter if there were staged before.
* **[-v]:** Puts the diff of your change in the editor so you can see exactly what changes you’re committing.
* **[-m]**: Allows us to add our comments in the command line.

## Tagging commits

**git tag -a <v#.#> [commit\_code] [-m ‘note’]:** Stores an annotated commit associated with our last commit with a note in the command line. It saves: tagger name, tagger email, tag date and tagging message. It can be signed and verified with GNU Privacy Guard (GPG).

**git tag -d <v#.#>:** Removes a tag in my local repository.

**git tag -l [’pattern’]:** Show all the tags in the current directory in alphabetical order.

**git show <v#.#>:** Shows information of a tag or commit.

**git push <remote> --follow-tags:** Transfers all annunciated tags to the remote server.

**git push <remote> --tags:** Transfers all tags to the remote server.

**git** **tag <remote> -delete <v#.#>:** Removes the tag from remote server.

## Checking commits

*git log [<options>] [<revision-range>] [[--] <path>…​]*

### Shaping commit report

**HEAD =** is a shortcut meaning "the most recent commit".

**HEAD~1 =** refers to the commit before the last.

**git show <hash>:** Shows information and git diff of the commit.

**git annotate <file>:** Shows who made the last change to each line of a file and when.

|  |  |
| --- | --- |
| **Command** | **Example** |
| **git log:** Lists all the info of each commit in the current branch.   * Press the space bar to go down a page * the 'q' key to quit. | Text, letter  Description automatically generated |
| **git log --pretty=oneline:** Shows one commit per line in the current branch. |  |
| **git log --pretty=format:"format\_code":** Shows one commit per line in the current branch with a custom information |  |
| **git log --pretty=format:" format\_code" --graph:** Adds a nice little ASCII graph showing your branch and merge history. |  |
| **git log --oneline --decorate --graph --all:** Prints out the history of your commits, showing where your branch pointers are  and how your history has diverged. |  |
| **git log -p:** Apply *git diff* for each two commits. | Text  Description automatically generated |
| **git log --stat:** Counts lines changes in each file tracked. |  |
| **--shortstat** | Display only the changed/insertions/deletions line from the --stat command. |
| **--name-only** | Show the list of files modified after the commit information. |
| **--name-status** | Show the list of files affected with added/modified/deleted information as well. |
| **--abbrev-commit** | Show only the first few characters of the SHA-1 checksum instead of all 40. |
| **--relative-date** | Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format. |

### Filtering commits

**git log --since=**2.weeks **or** "2008-01-15" **or** "2 years 1 day 3 minutes ago"

**git log --until=**2.weeks **or** "2008-01-15"

**git log <-#>:** We can limit the number of commits to show with -#.

**git log --author name:** Option allows you to filter on a specific author. You can specify more than one instance

**git log --grep pattern:** Lets you search for keywords in the commit messages. You can specify more than one instance. Adding **--all-match** will get the commits with all matches.

git log --grep camp --grep data --author Angel

git log --grep camp --grep data --all-match

**git log -S function\_name:** Takes a string and shows only those commits that changed the number of occurrences of that string.

**git log -- path/to/file:** This is always the *last option* and is generally preceded by double dashes. It limits the log output to commits that introduced a change to those files.

### Specifiers for --pretty=format option

*Table 1. Useful specifiers for* git log --pretty=format

|  |  |
| --- | --- |
| **Specifier** | **Description of Output** |
| %H | Commit hash |
| %h | Abbreviated commit hash |
| %T | Tree hash |
| %t | Abbreviated tree hash |
| %P | Parent hashes |
| %p | Abbreviated parent hashes |
| %an | Author name |
| %ae | Author email |
| %ad | Author date (format respects the --date=option) |
| %ar | Author date, relative |
| %cn | Committer name |
| %ce | Committer email |
| %cd | Committer date |
| %cr | Committer date, relative |
| %s | Subject |

## Removing, moving and renaming files

**git rm file\_name:** Removes the file from my local computer and add that change to the staging area if the file is committed.

**git rm log/\\*.log:** Removes all “log/\*.log”, so we need to use \\* before any regular expression.

**git rm -f file\_name:** Removes files from the staging area.

**git rm --cached file\_name:** Removes files from the staging area but keep files in local folder.

**git clean -f:** Deletes untracked files.

**git mv old\_file\_name new\_file\_name:** Rename a file.

## Undoing Things

**git commit --amend:** If you want to redo that commit, make the additional changes you forgot, stage them, and commit again. This command takes your staging area and uses it for the commit. *Only amend commits that are still local.*

### Restore an old version of a file

**git checkout <hash> <filename>**

### Undo changes to unstaged files

**git checkout -- <filename>** orgit restore <file>

### Undo changes to staged files

**git reset HEAD <filename>** or git restore --staged <file>

**git checkout -- <filename>** or git restore <file>

### Undo all changes

**git reset**

**git checkout -- .**

## Creating custom functions

**git config --global alias.<function> ‘code’**

git config --global alias.unstage ‘reset HEAD --'

git config --global alias.last ‘log -1 HEAD'

## Working with branches

### Creating branches

**git checkout -b <branch\_name>:** Creates a new branch with the information of current branch and move git to that branch.

**git switch -c <branch\_name>:** The same.

### Switching branches

**git checkout <branch\_name>:** Move git to that branch.

**git switch <branch\_name>:** The same

### Getting branch info

**git branch:** Show the list of branches in the current repository with an \* in the current branch.

**git diff branch-1..branch-2:** Shows the difference between two branches.

### Merging branches

**git merge <source\_branch> [destination(current)\_branch]:** Add the named branch to your current branch. When we are working in branch and some make a change in master we should merge in our current branch before continuing working.

### Solving conflicts

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a video game

Description automatically generated with medium confidence

Text

Description automatically generated

Note that <<<<<<<, =======, and >>>>>>> lines have been completely removed.

A screenshot of a computer

Description automatically generated with medium confidence

### Delete a branch

**git branch -d <branch\_name>:** After merging a branch into other we can delete that branch. If we don’t do it, we would be duplicating the information in our git.

### Other things

**git checkout -b <branch\_name> <remote>/<branch\_name>:** Creates a new branch from a fetch remote one.

**git checkout --track <remote>/<branch\_name>:** The same.

**git checkout <branch\_name>:** The same if we have a branch name no repeated in any other remote.

**git branch <branch\_name>:** Just create a new branch with the current branch information.

**git fetch --all:** Git never goes to any server to check so we need to update all our remote branches before getting information about any situation.

**git branch [-v] [--merged] [branch\_name] [--all]:** List all the existing branches.

* **[-v]:** Shows the last commit of each branch.
* **[--merged or --no-merged]:** Lists branches that you have merged into the branch you’re currently on or the one specified in with branch\_name argument. So you can delete the branch without the \*.
* **[--all]:** Show rename branch changes

Text

Description automatically generated

* iss53 branch is tracking origin/iss53 and is **“ahead”** by two, meaning that we have two commits locally that are not pushed to the server.
* origin/master and is up to date.
* serverfix branch is tracking the server-fix-good branch on our teamone server and is **ahead** by three and **behind** by one, meaning that there is one commit on the server we haven’t merged in yet and three commits locally that we haven’t pushed.
* testing branch is not tracking any remote branch.

**git branch [--move <bad-branch-name> <corrected-branch-name>]:** Rename a branch.

* Do not rename branches that are still in use by other collaborators
* Do not rename a branch like master/main/mainline

**git push --set-upstream <remote> <corrected-branch-name >**

**git push <remote> --delete <bad-branch-name>**

To let others see the corrected branch on the remote.

## Working with Remotes

### Creating a remote

**git remote add <remote\_name> <url>:** To add a new remote git repository as a shortname. *shortname*’s master branch is now accessible locally.

**git init --bare:** Make the current working directory a remote where we can push changes.

### Consulting remotes

**git remote [-v]:** Shows remote and -v option lists the URLs related with each remote.

**git remote show <remote>:** Shows a full list of remote references.

**git ls-remote <remote>:** The same.

### Editing remotes

**git remote rename <old\_remote> <new\_remote>**

**git remote rm <remote-name>:** Removes all tracking branches and configuration settings associated with that remote.

**git remote remove <remote>:** The same.

**git tag <remote> -delete < v#.#>:**  Removes a tag from a remove directory.

**git push <remote> --delete <branch>:** Delete a branch in a server.

### Getting info from remotes

**git fetch <remote> <branch\_name>:** Goes out to that remote project and pulls down all the data from that remote project that you don’t have yet. After you do this, you should have references to all the branches from that remote, which you can merge in or inspect at any time. it doesn’t automatically merge it with any of your work or modify what you’re currently working on.

|  |  |
| --- | --- |
| Diagram  Description automatically generated |  |

**git pull** **<remote> <branch\_name>:** Automatically fetch and then merge that remote branch into your current branch (the code you’re currently working on).

**git config --global pull.rebase <"false" or "true">**: Rebase is an action in Git that allows you to rewrite commits from one branch onto another branch. Essentially, Git is deleting commits from one branch and adding them onto another.

### Sharing into to remotes

**git push <remote> [v#.# | --tags | --follow-tags] <branch\_name>:** Takes our last commit and put it in a remote repository or a tag specified. --tags will push all tags and --follow-tags will transfer just announced tags.

git push origin master

**git branch --set-upstream-to <remote>/<my\_remote\_branch> <my\_local\_branch>:** Push a new branch with a different name by keeping

**git push <remote> <my\_local\_branch>:<my\_remote\_branch>**