Git

Welcome to our Git cheatsheet! Git is a powerful and widely-used version control system that allows developers to efficiently track and manage changes to their code. Whether you're new to Git or a seasoned pro, this cheatsheet will provide a quick reference for some of the most commonly used Git commands and workflow. From initializing a repository and committing changes, to branching and merging, we'll cover the basics to help you get started with Git or to refresh your memory on some of its key features. Let's dive in and start mastering Git!

Init

This command initializes a new Git repository in the current directory

- git config --global init.defaultBranch

Using 'master' as the name for the initial branch. This default branch name is subject to change. To configure the initial branch name to use in all of your new repositories

Info

Here are a few examples of how to use git config to set your name and email:

Global

- git config --global user.name "Your Name": This command sets your name globally, which means it will be used for all Git repositories on your computer.
- git config --global user.email "your_email@example.com": This command sets your email globally, which means it will be used for all Git repositories on your computer.

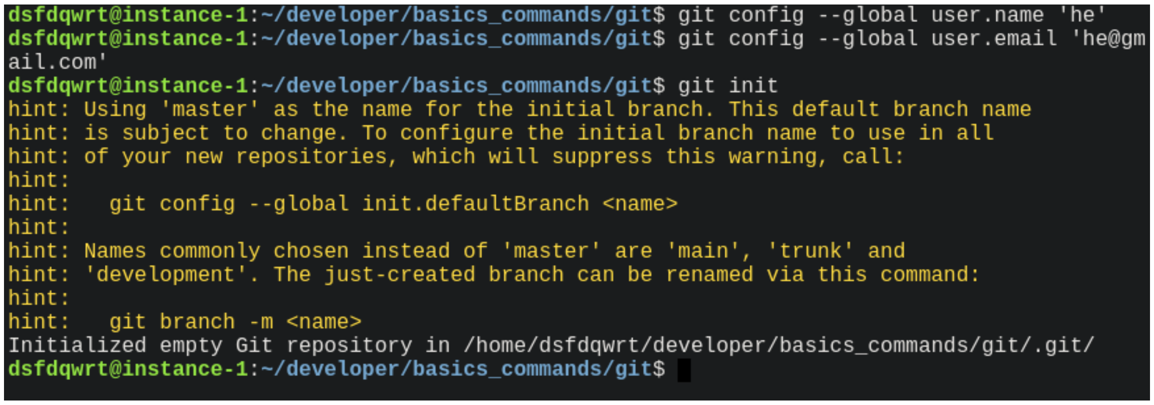
Current file

- git config user.name "Your Name": This command sets your name for the current repository only.
- git config user.email "your_email@example.com": This command sets your email for the current repository only.

It's important to note that the global flag --global is optional, if you don't use it the configuration will be set for the current repository only.

You can check your configuration by running: *git config --list*.

It's a good practice to set your name and email correctly before making your first commit, because they will be associated with all commits you make in that repository.



```
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git config --global user.name 'he'
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git config --global user.email 'he@gm
ail.com'
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/dsfdqwrt/developer/basics_commands/git/.git/
dsfdqwrt@instance-1:~/developer/basics_commands/git$ 
```

Add

git add : This command stages changes to the specified file, which means it prepares the file to be included in the next commit.

Commit

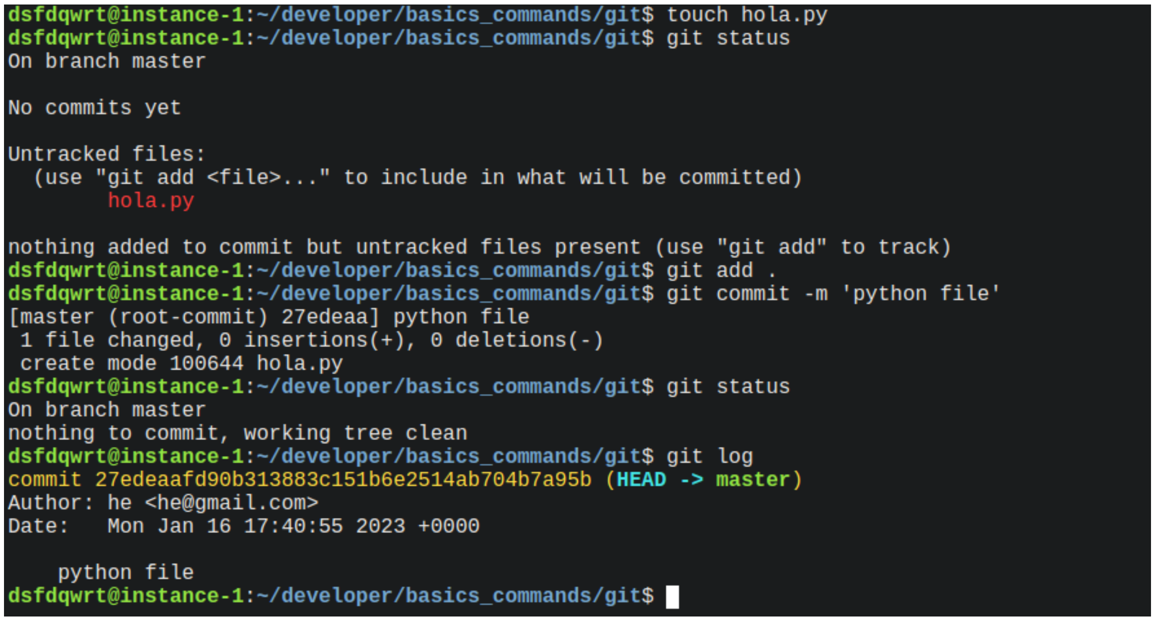
git commit -m "": This command creates a new commit, which is a snapshot of the project's files at a specific point in time. The -m option allows you to include a message describing the changes made in the commit.

Status

git status: This command shows the status of the repository, including any changes that have been made but not yet committed.

Log

git log: This command shows a log of all the commits in the repository, including their author, date, and commit message.



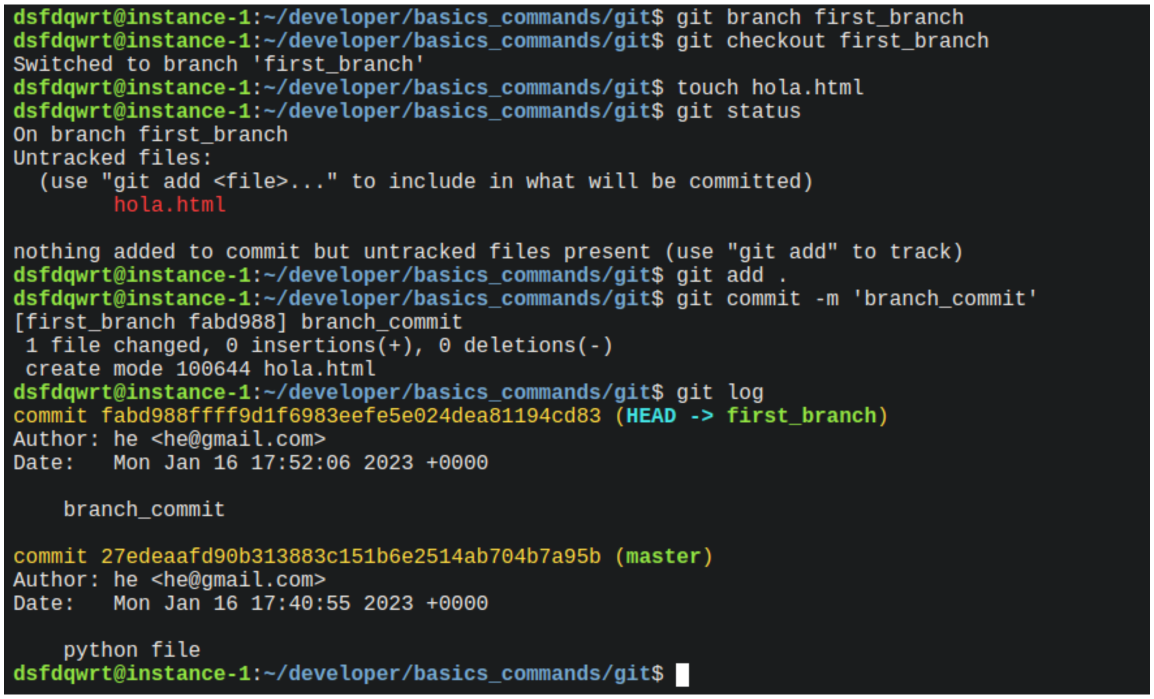
```
dsfdqwrt@instance-1:~/developer/basics_commands/git$ touch hola.py
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hola.py

nothing added to commit but untracked files present (use "git add" to track)
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git add .
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git commit -m 'python file'
[master (root-commit) 27edeaa] python file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hola.py
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git status
On branch master
nothing to commit, working tree clean
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git log
commit 27edeaafd90b313883c151b6e2514ab704b7a95b (HEAD -> master)
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:40:55 2023 +0000

    python file
dsfdqwrt@instance-1:~/developer/basics_commands/git$
```

Branch

git branch <branch_name>

Branches are used to work on multiple versions of the project simultaneously. By default, when you create a new Git repository, it has a single branch called master.

- You can create a new branch using the command *git branch <branch_name>*.
- Once a new branch is created you can switch to that branch using *git checkout <branch_name>*. This allows you to make changes to the branch without affecting the master branch.



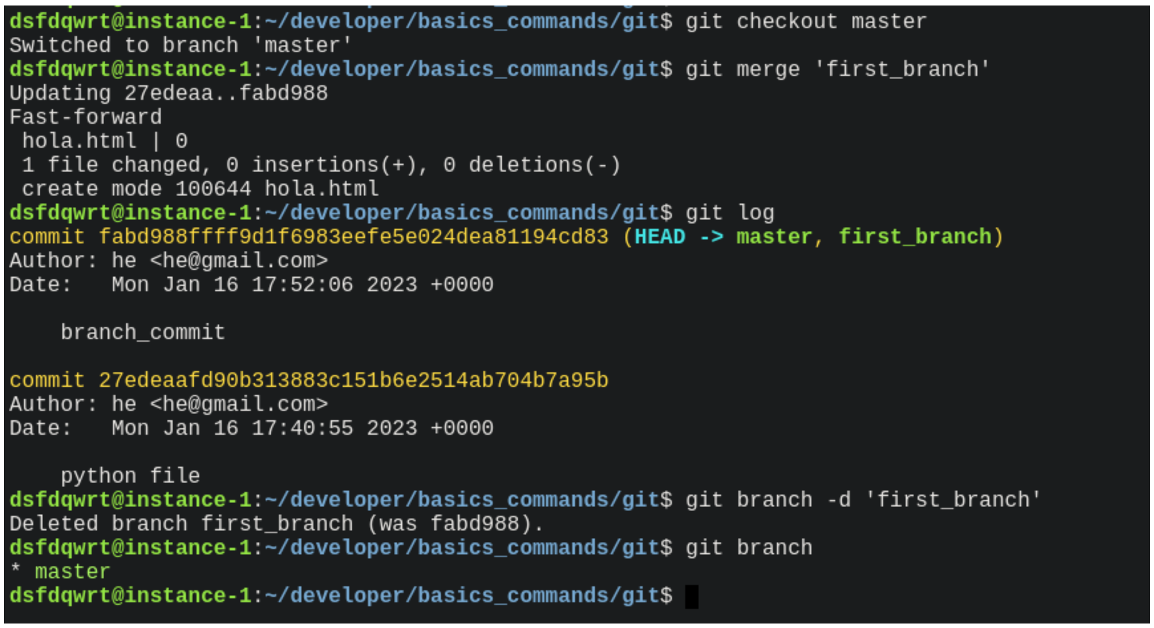
```
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git branch first_branch
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git checkout first_branch
Switched to branch 'first_branch'
dsfdqwrt@instance-1:~/developer/basics_commands/git$ touch hola.html
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git status
On branch first_branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hola.html

nothing added to commit but untracked files present (use "git add" to track)
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git add .
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git commit -m 'branch_commit'
[first_branch fabd988] branch_commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hola.html
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git log
commit fabd988ffff9d1f6983eefe5e024dea81194cd83 (HEAD -> first_branch)
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:52:06 2023 +0000

    branch_commit

commit 27edeaafd90b313883c151b6e2514ab704b7a95b (master)
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:40:55 2023 +0000

    python file
dsfdqwrt@instance-1:~/developer/basics_commands/git$
```

Git merge <branch_name>

When you are done with your changes, you can merge the branch back into the master branch using the command

- *git merge <branch_name>*. Once you merge the branch, you can delete it using *git branch -d <branch_name>*.

It is a best practice to create a new branch for each feature you're working on, so that you can work on multiple features simultaneously without affecting the master branch. This also allows you to easily switch between different features and merge them back into the master branch when they're done.



```
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git checkout master
Switched to branch 'master'
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git merge 'first_branch'
Updating 27edeaa..fabd988
Fast-forward
 hola.html | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hola.html
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git log
commit fabd988ffff9d1f6983eefe5e024dea81194cd83 (HEAD -> master, first_branch)
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:52:06 2023 +0000

    branch_commit

commit 27edeaafd90b313883c151b6e2514ab704b7a95b
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:40:55 2023 +0000

    python file
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git branch -d 'first_branch'
Deleted branch first_branch (was fabd988).
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git branch
* master
dsfdqwrt@instance-1:~/developer/basics_commands/git$ 
```

Rebase

The git rebase command is used to reapply commits on top of another base tip. It can be used to synchronize a branch with its upstream branch, or to clean up the commit history of a branch by squashing or reordering commits.

- git rebase : This command synchronizes the current branch with its upstream branch. It reapplies the commits in the current branch on top of the tip of the upstream branch. This can be useful to keep your branch up-to-date with the latest changes made in the upstream branch.
- git rebase -i : This command opens an interactive rebase session, which allows you to modify the commit history of the current branch. You can use commands like pick, squash, reword, edit, and drop to reorder, combine, or remove commits.

It's important to note that git rebase modifies the commit history, it creates new commits that replaces the old ones, so it's not recommended to use it when other people are working on the same branch or the branch is public.

It's also important to note that if you're rebasing a branch that you've pushed to a remote repository, you should use git pull --rebase instead of git pull to avoid creating merge commits.

Forks

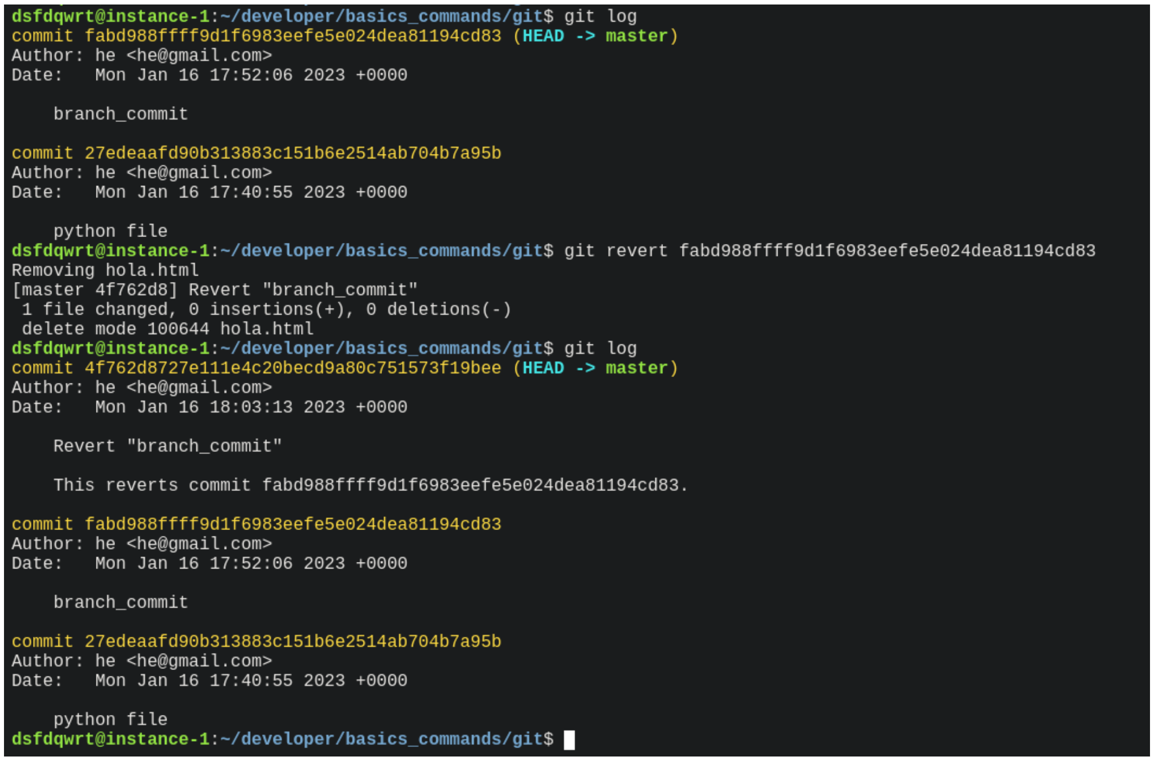
A fork is a copy of a repository that you own. You can make changes to your fork without affecting the original repository.

Git is a powerful and widely-used version control system that allows developers to efficiently track and manage changes to their code. However, as with any powerful tool, sometimes things can go wrong and you may need to undo changes. We'll be covering three of the most commonly used Git commands for undoing changes: revert, reset, and checkout.

Revert

Revert

- git revert <commit_hash>: This command creates a new commit that undoes the changes made in the specified commit. The original commit will remain in the commit history, but the changes it made will be undone.



```
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git log
commit fabd988ffff9d1f6983eefe5e024dea81194cd83 (HEAD -> master)
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:52:06 2023 +0000

    branch_commit

commit 27edeaafd90b313883c151b6e2514ab704b7a95b
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:40:55 2023 +0000

    python file
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git revert fabd988ffff9d1f6983eefe5e024dea81194cd83
Removing hola.html
[master 4f762d8] Revert "branch_commit"
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 hola.html
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git log
commit 4f762d8727e111e4c20becd9a80c751573f19bee (HEAD -> master)
Author: he <he@gmail.com>
Date:   Mon Jan 16 18:03:13 2023 +0000

    Revert "branch_commit"

    This reverts commit fabd988ffff9d1f6983eefe5e024dea81194cd83.

commit fabd988ffff9d1f6983eefe5e024dea81194cd83
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:52:06 2023 +0000

    branch_commit

commit 27edeaafd90b313883c151b6e2514ab704b7a95b
Author: he <he@gmail.com>
Date:   Mon Jan 16 17:40:55 2023 +0000

    python file
dsfdqwrt@instance-1:~/developer/basics_commands/git$ 
```

Revert is a command in Git that allows you to undo a specific commit, while keeping a record of it in the commit history. This is a preferred method of undoing changes in Git because it allows you to maintain a clear and accurate history of your codebase.

When you use the revert command, Git creates a new commit that undoes the changes made in the commit you are reverting. This new commit has its own unique hash, and it becomes part of the commit history. This allows you to see the changes that were made, and the changes that were undone, providing a clear picture of the development process.

In contrast, other undo commands like reset and checkout discards the commits completely and make them unrecoverable, which can lead to confusion and loss of important information.

Additionally, using revert allows other team members to see the changes that were made and undone, which can be helpful for collaboration and understanding how the codebase has evolved over time.

In summary, using revert instead of other undo commands allows for a clear and accurate history of the development process, and allows for better collaboration within the team.

Reset

- git reset <commit_hash>: This command undoes commits by moving the HEAD and current branch pointer to the specified commit. It discards commits made after the specified commit. This command also discards the changes made in the working directory.

Checkout

- git checkout : This command discards changes made to a specific file.

Remote

Git is a distributed version control system, which means that multiple copies of the codebase can be stored in different locations, called "remotes". The git remote command allows you to manage these remote repositories.

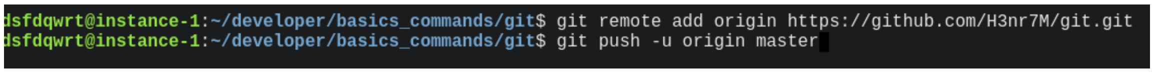
- View a list of existing remotes: git remote -v
- Add a new remote: git remote add
- Rename an existing remote: git remote rename <old_name> <new_name>
- Remove a remote: git remote remove

Pull

The git pull command is used to fetch the changes from a remote repository and merge them into the local copy of the codebase. The basic syntax for a pull is git pull

Push

The git push command is used to send the changes from the local copy of the codebase to a remote repository. The basic syntax for a push is git push



```
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git remote add origin https://github.com/H3nr7M/git.git
dsfdqwrt@instance-1:~/developer/basics_commands/git$ git push -u origin master
```

In conclusion, this Git cheatsheet has covered some of the most commonly used Git commands and workflow, providing a quick reference for developers to efficiently track and manage changes to their code. We've covered basic Git commands like initializing a repository, committing changes, branching, merging and also some advanced commands like remote, pull and push.

We've also discussed the importance of undo commands like revert and explained why it's a preferred method of undoing changes in Git, by keeping a clear and accurate history of your codebase.

Remember that Git is a powerful tool and like any tool, it takes practice to master it. We encourage you to keep this cheatsheet handy as a reference and to continue practicing Git commands and workflow. With time and practice, you'll become a proficient Git user and be able to navigate and manage your codebase with ease.