Git Cheat Sheet

01 Git configuration

```
$ git config --global user.name “Your Name”
```

Set the name that will be attached to your commits and tags.

```
$ git config --global user.email “you@example.com”
```

Set the e-mail address that will be attached to your commits and tags.

```
$ git config --global color.ui auto
```

Enable some colorization of Git output.

02 Starting A Project

```
$ git init [project name]
```

Create a new local repository. If **[project name]** is provided, Git will create a new directory name **[project name]** and will initialize a repository inside it. If **[project name]** is not provided, then a new repository is initialized in the current directory.

```
$ git clone [project url]
```

Downloads a project with the entire history from the remote repository.

03 Day-To-Day Work

```
$ git status
```

Displays the status of your working directory. Options include new, staged, and modified files. It will retrieve branch name, current commit identifier, and changes pending commit.

```
$ git add [file]
```

Add a file to the **staging** area. Use in place of the full file path to add all changed files from the **current directory** down into the **directory tree**.

```
$ git diff [file]
```

Show changes between **working directory** and **staging area**.

```
$ git diff --staged [file]
```

Shows any changes between the **staging area** and the **repository**.

```
$ git checkout -- [file]
```

Discard changes in **working directory**. This operation is **unrecoverable**.

```
$ git reset [file]
```

Revert your **repository** to a previous known working state.

```
$ git commit
```

Create a new **commit** from changes added to the **staging area**.
The **commit** must have a message!

```
$ git rm [file]
```

Remove file from **working directory** and **staging area**.

```
$ git stash
```

Put current changes in your **working directory** into **stash** for later use.

```
$ git stash pop
```

Apply stored **stash** content into **working directory**, and clear **stash**.

```
$ git stash drop
```

Delete a specific **stash** from all your previous **stashes**.

04 Git branching model

```
$ git branch [-a]
```

List all local branches in repository. With **-a**: show all branches (with remote).

```
$ git branch [branch_name]
```

Create new branch, referencing the current **HEAD**.

```
$ git checkout [-b][branch_name]
```

Switch **working directory** to the specified branch. With **-b**: Git will create the specified branch if it does not exist.

```
$ git merge [from name]
```

Join specified **[from name]** branch into your current branch (the one you are on currently).

```
$ git branch -d [name]
```

Remove selected branch, if it is already merged into any other.
-D instead of **-d** forces deletion.

05 Review your work

```
$ git log [-n count]
```

List commit history of current branch. **-n count** limits list to last **n** commits.

```
$ git log --oneline --graph --decorate
```

An overview with reference labels and history graph. One commit per line.

```
$ git log ref..
```

List commits that are present on the current branch and not merged into **ref**. A **ref** can be a branch name or a tag name.

```
$ git log ..ref
```

List commit that are present on **ref** and not merged into current branch.

```
$ git reflog
```

List operations (e.g. checkouts or commits) made on local repository.

06 Tagging known commits

```
$ git tag
```

List all tags.

```
$ git tag [name] [commit sha]
```

Create a tag reference named **name** for current commit. Add **commit sha** to tag a specific commit instead of current one.

```
$ git tag -a [name] [commit sha]
```

Create a tag object named **name** for current commit.

```
$ git tag -d [name]
```

Remove a tag from local repository.

07 Reverting changes

```
$ git reset [--hard] [target reference]
```

Switches the current branch to the **target reference**, leaving a difference as an uncommitted change. When **--hard** is used, all changes are discarded.

```
$ git revert [commit sha]
```

Create a new commit, reverting changes from the specified commit. It generates an **inversion** of changes.

08 Synchronizing repositories

```
$ git fetch [remote]
```

Fetch changes from the **remote**, but not update tracking branches.

```
$ git fetch --prune [remote]
```

Delete remote Refs that were removed from the **remote** repository.

```
$ git pull [remote]
```

Fetch changes from the **remote** and merge current branch with its upstream.

```
$ git push [--tags] [remote]
```

Push local changes to the **remote**. Use **--tags** to push tags.

```
$ git push -u [remote] [branch]
```

Push local branch to **remote** repository. Set its copy as an upstream.

Commit	an object
Branch	a reference to a commit; can have a **tracked upstream**
Tag	a reference (standard) or an object (annotated)
Head	a place where your **working directory** is now

A Git installation

For GNU/Linux distributions, Git should be available in the standard system repository. For example, in Debian/Ubuntu please type in the **terminal**:

```
$ sudo apt-get install git
```

If you need to install Git from source, you can get it from git-scm.com/downloads.

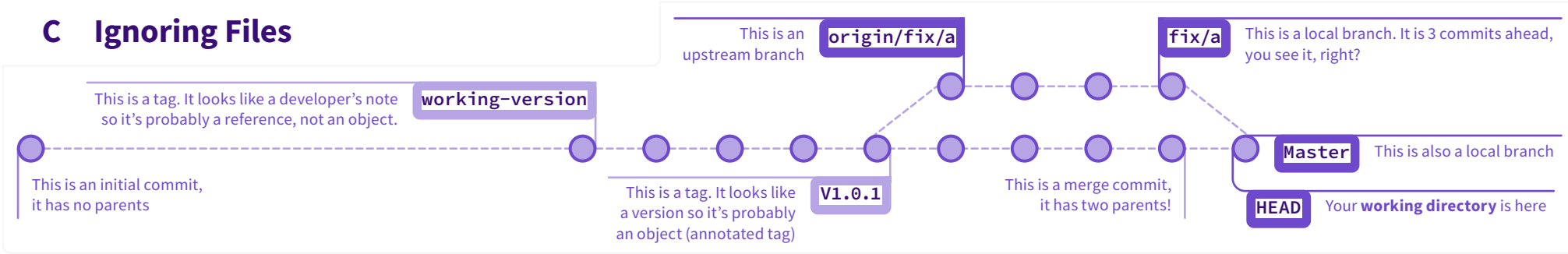
An excellent Git course can be found in the great **Pro Git** book by Scott Chacon and Ben Straub. The book is available online for free at git-scm.com/book.

B Ignoring Files

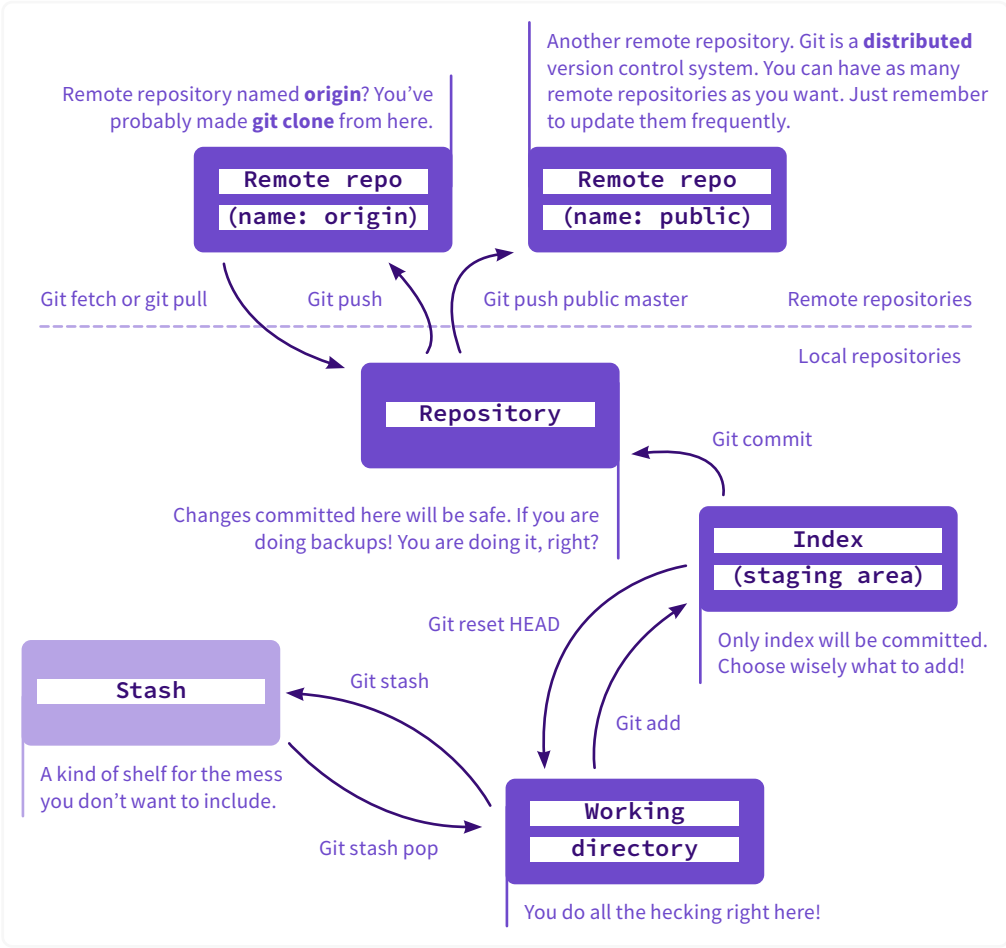
```
$ cat .gitignore
/logs/*
!logs/.gitkeep
/tmp
*.swp
```

Verify the .gitignore file exists in your project and ignore certain type of files, such as all files in **logs** directory (excluding the **.gitkeep** file), whole **tmp** directory and all files ***.swp**. File ignoring will work for the directory (and children directories) where **.gitignore** file is placed.

C Ignoring Files



D The zoo of working areas



GitHub

Git Cheat Sheet

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

Install

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git for All Platforms
http://git-scm.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Configure tooling

Configure user information for all local repositories

`$ git config --global user.name "[name]"`
Sets the name you want attached to your commit transactions

`$ git config --global user.email "[email address]"`
Sets the email you want attached to your commit transactions

`$ git config --global color.ui auto`
Enables helpful colorization of command line output

Create repositories

When starting out with a new repository, you only need to do it once; either locally, then push to GitHub, or by cloning an existing repository.

`$ git init`
Turn an existing directory into a git repository

`$ git clone [url]`
Clone (download) a repository that already exists on GitHub, including all of the files, branches, and commits

The .gitignore file

Sometimes it may be a good idea to exclude files from being tracked with Git. This is typically done in a special file named `.gitignore`. You can find helpful templates for `.gitignore` files at github.com/github/gitignore.

Branches

Branches are an important part of working with Git. Any commits you make will be made on the branch you're currently "checked out" to. Use `git status` to see which branch that is.

`$ git branch [branch-name]`
Creates a new branch

`$ git checkout [branch-name]`
Switches to the specified branch and updates the working directory

`$ git merge [branch]`
Combines the specified branch's history into the current branch. This is usually done in pull requests, but is an important Git operation.

`$ git branch -d [branch-name]`
Deletes the specified branch

Synchronize changes

Synchronize your local repository with the remote repository on GitHub.com

`$ git fetch`
Downloads all history from the remote tracking branches

`$ git merge`
Combines remote tracking branch into current local branch

`$ git push`
Uploads all local branch commits to GitHub

`$ git pull`
Updates your current local working branch with all new commits from the corresponding remote branch on GitHub. `git pull` is a combination of `git fetch` and `git merge`

GitHub Git Cheat Sheet

Make changes

Browse and inspect the evolution of project files

`$ git log`
Lists version history for the current branch

`$ git log --follow [file]`
Lists version history for a file, including renames

`$ git diff [first-branch]...[second-branch]`
Shows content differences between two branches

`$ git show [commit]`
Outputs metadata and content changes of the specified commit

`$ git add [file]`
Snapshots the file in preparation for versioning

`$ git commit -m "[descriptive message]"`
Records file snapshots permanently in version history

Redo commits

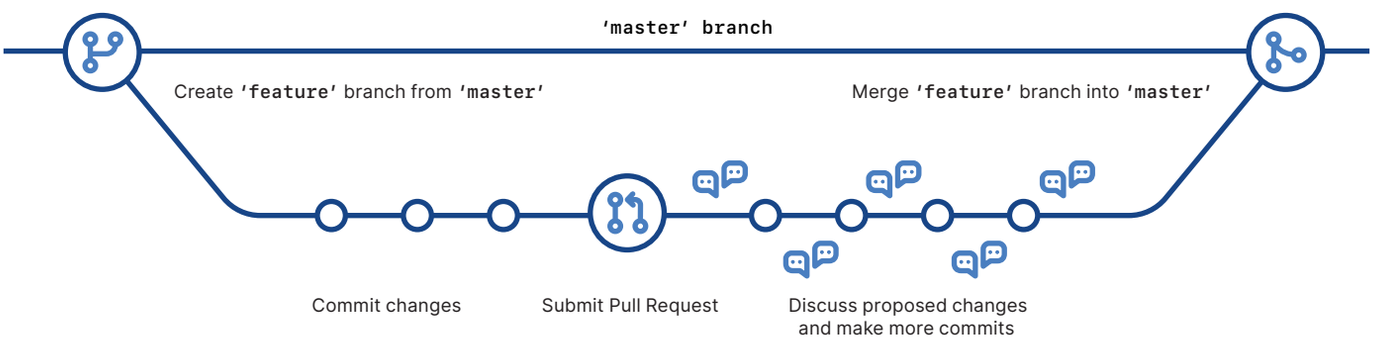
Erase mistakes and craft replacement history

`$ git reset [commit]`
Undoes all commits after [commit], preserving changes locally

`$ git reset --hard [commit]`
Discards all history and changes back to the specified commit

CAUTION! Changing history can have nasty side effects. If you need to change commits that exist on GitHub (the remote), proceed with caution. If you need help, reach out at github.community or contact support.

GitHub Flow



Glossary

`git:` an open source, distributed version-control system
`GitHub:` a platform for hosting and collaborating on Git repositories
`commit:` a Git object, a snapshot of your entire repository compressed into a SHA
`branch:` a lightweight movable pointer to a commit
`clone:` a local version of a repository, including all commits and branches
`remote:` a common repository on GitHub that all team member use to exchange their changes
`fork:` a copy of a repository on GitHub owned by a different user
`pull request:` a place to compare and discuss the differences introduced on a branch with reviews, comments, integrated tests, and more
`HEAD:` representing your current working directory, the HEAD pointer can be moved to different branches, tags, or commits when using `git checkout`

Git Cheat Sheet

GIT BASICS

`git init <directory>`	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
`git clone <repo>`	Clone repo located at `<repo>` onto local machine. Original repo can be located on the local filesystem or on a remote machine via `HTTP` or `SSH`.
`git config user.name <name>`	Define author name to be used for all commits in current repo. Devs commonly use `--global` flag to set config options for current user.
`git add <directory>`	Stage all changes in `<directory>` for the next commit. Replace `<directory>` with a `<file>` to change a specific file.
`git commit -m "<message>"`	Commit the staged snapshot, but instead of launching a text editor, use `<message>` as the commit message.
`git status`	List which files are staged, unstaged, and untracked.
`git log`	Display the entire commit history using the default format. For customization see additional options.
`git diff`	Show unstaged changes between your index and working directory.

UNDOING CHANGES

`git revert <commit>`	Create new commit that undoes all of the changes made in `<commit>`, then apply it to the current branch.
`git reset <file>`	Remove `<file>` from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
`git clean -n`	Shows which files would be removed from working directory. Use the `-f` flag in place of the `-n` flag to execute the clean.

REWRITING GIT HISTORY

`git commit --amend`	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
`git rebase <base>`	Rebase the current branch onto `<base>`. `<base>` can be a commit ID, branch name, a tag, or a relative reference to `HEAD`.
`git reflog`	Show a log of changes to the local repository's `HEAD`. Add `--relative-date` flag to show date info or `--all` to show all refs.

GIT BRANCHES

`git branch`	List all of the branches in your repo. Add a `<branch>` argument to create a new branch with the name `<branch>`.
`git checkout -b <branch>`	Create and check out a new branch named `<branch>`. Drop the `-b` flag to checkout an existing branch.
`git merge <branch>`	Merge `<branch>` into the current branch.

REMOTE REPOSITORIES

`git remote add <name> <url>`	Create a new connection to a remote repo. After adding a remote, you can use `<name>` as a shortcut for `<url>` in other commands.
`git fetch <remote> <branch>`	Fetches a specific `<branch>`, from the repo. Leave off `<branch>` to fetch all remote refs.
`git pull <remote>`	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
`git push <remote> <branch>`	Push the branch to `<remote>`, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

Additional Options +

GIT CONFIG

`git config --global user.name <name>`	Define the author name to be used for all commits by the current user.
`git config --global user.email <email>`	Define the author email to be used for all commits by the current user.
`git config --global alias. <alias-name> <git-command>`	Create shortcut for a Git command. E.g. `alias.glog "log --graph --oneline"` will set `"git glog"` equivalent to `"git log --graph --oneline.`
`git config --system core.editor <editor>`	Set text editor used by commands for all users on the machine. `<editor>` arg should be the command that launches the desired editor (e.g., vi).
`git config --global --edit`	Open the global configuration file in a text editor for manual editing.

GIT LOG

`git log -<limit>`	Limit number of commits by `<limit>`. E.g. `"git log -5"` will limit to 5 commits.
`git log --oneline`	Condense each commit to a single line.
`git log -p`	Display the full diff of each commit.
`git log --stat`	Include which files were altered and the relative number of lines that were added or deleted from each of them.
`git log --author= "<pattern>"`	Search for commits by a particular author.
`git log --grep="<pattern>"`	Search for commits with a commit message that matches `<pattern>`.
`git log <since>..<until>`	Show commits that occur between `<since>` and `<until>`. Args can be a commit ID, branch name, `HEAD`, or any other kind of revision reference.
`git log -- <file>`	Only display commits that have the specified file.
`git log --graph --decorate`	`--graph` flag draws a text based graph of commits on left side of commit msgs. `--decorate` adds names of branches or tags of commits shown.

GIT DIFF

`git diff HEAD`	Show difference between working directory and last commit.
`git diff --cached`	Show difference between staged changes and last commit

GIT RESET

`git reset`	Reset staging area to match most recent commit, but leave the working directory unchanged.
`git reset --hard`	Reset staging area and working directory to match most recent commit and **overwrites all changes** in the working directory.
`git reset <commit>`	Move the current branch tip backward to `<commit>`, reset the staging area to match, but leave the working directory alone.
`git reset --hard <commit>`	Same as previous, but resets both the staging area & working directory to match. **Deletes** uncommitted changes, and **all commits after** `<commit>`.

GIT REBASE

`git rebase -i <base>`	Interactively rebase current branch onto `<base>`. Launches editor to enter commands for how each commit will be transferred to the new base.

GIT PULL

`git pull --rebase <remote>`	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.

GIT PUSH

`git push <remote> --force`	Forces the `git push` even if it results in a non-fast-forward merge. Do not use the `--force` flag unless you're absolutely sure you know what you're doing.
`git push <remote> --all`	Push all of your local branches to the specified remote.
`git push <remote> --tags`	Tags aren't automatically pushed when you push a branch or use the `--all` flag. The `--tags` flag sends all of your local tags to the remote repo.

GitHub
GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

For Linux and Solaris platforms, the latest release is available on the official Git web site.

Git for All Platforms
http://git-scm.com

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`
set a name that is identifiable for credit when review version history
`git config --global user.email "[valid-email]"`
set an email address that will be associated with each history marker
`git config --global color.ui auto`
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`
initialize an existing directory as a Git repository
`git clone [url]`
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`
diff of what is staged but not yet committed
`git commit -m "[descriptive message]"`
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`
list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`
create a new branch at the current commit
`git checkout`
switch to another branch and check it out into your working directory
`git merge [branch]`
merge the specified branch's history into the current one
`git log`
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`	show the commit history for the currently active branch
`git log branchB..branchA`	show the commits on branchA that are not on branchB
`git log --follow [file]`	show the commits that changed file, even across renames
`git diff branchB...branchA`	show the diff of what is in branchA that is not in branchB
`git show [SHA]`	show any object in Git in human-readable format

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`	add a git URL as an alias
`git fetch [alias]`	fetch down all the branches from that Git remote
`git merge [alias]/[branch]`	merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`	Transmit local branch commits to the remote repository branch
`git pull`	fetch and merge any commits from the tracking remote branch

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`	delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`	change an existing file path and stage the move
`git log --stat -M`	show all commit logs with indication of any paths that moved

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`	apply any commits of current branch ahead of specified one
`git reset --hard [commit]`	clear staging area, rewrite working tree from specified commit

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`	system wide ignore pattern for all local repositories

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`	Save modified and staged changes
`git stash list`	list stack-order of stashed file changes
`git stash pop`	write working from top of stash stack
`git stash drop`	discard the changes from top of stash stack

Branches are one of the core concepts in Git. And there's an endless amount of things you can do with them. You can create and delete them, rename and publish them, switch and compare them... and so much more.

My intention with this post is to create a comprehensive overview of the things you can do with branches in Git. I didn't want to

- How to create branches
- How to rename branches
- How to switch branches
- How to publish branches
- How to track branches
- How to delete branches
- How to merge branches
- How to rebase branches
- How to compare branches

How to Create a Branch in Git

Before you can work with branches, you need to have some in your repository. So let's start by talking about how to create branches:

```
$ git branch <new-branch-name>
```

When providing just a name to the `git branch` command, Git will assume that you want to start your new branch based on your currently checked out revision. If you'd like your new branch to start at a *specific* revision, you can simply add the revision's SHA-1 hash:

```
$ git branch <new-branch-name> 89a2faad
```

happens by publishing an existing local branch – which we'll talk about later.

How to Rename a Branch in Git

Mistyping a branch's name or simply changing your mind after the fact is all too easy. That's why Git makes it pretty easy to rename a local branch. If you want to rename your current HEAD branch, you can use the following command:

```
$ git branch -m <new-name>
```

In case you'd like to rename a different local branch (which is NOT currently checked out), you'll have to provide the old *and* the new name:

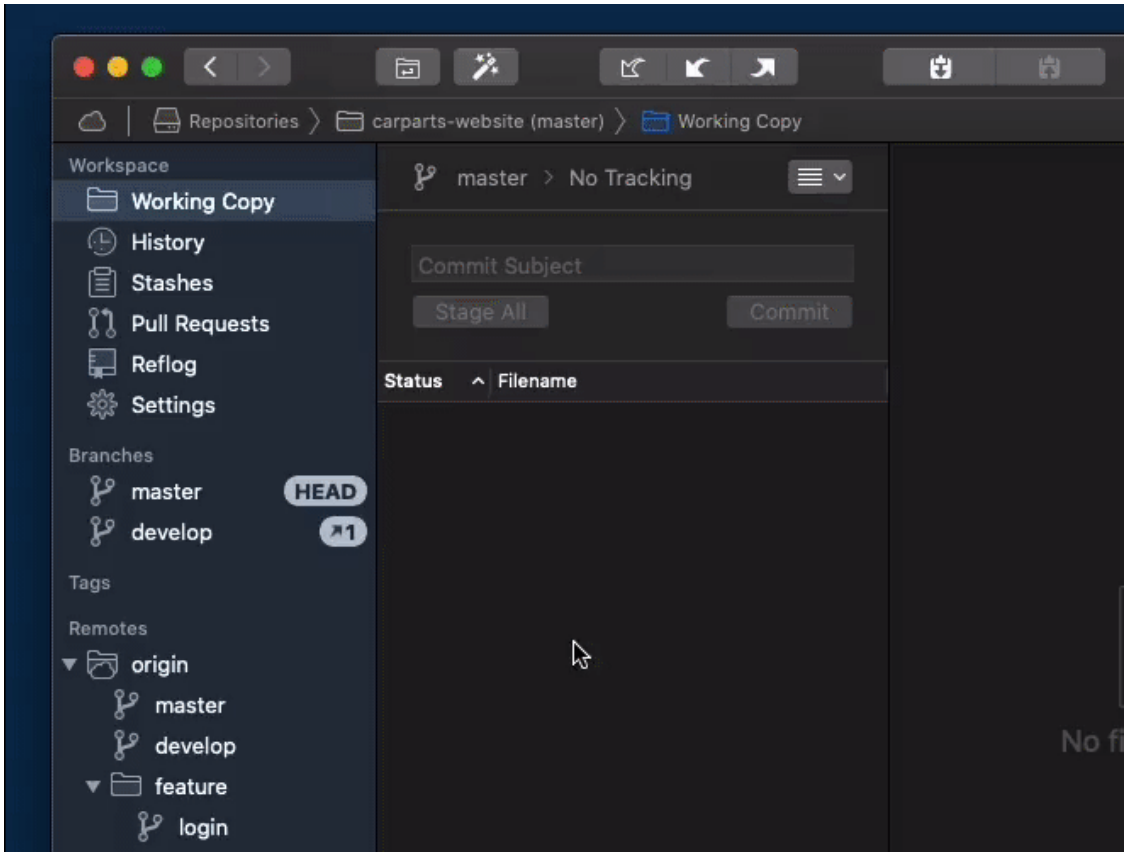
```
$ git branch -m <old-name> <new-name>
```

These commands, again, are used to work with local branches. If you'd like to rename a remote branch, things are a little bit more complicated - because Git doesn't allow you to rename remote branches.

In practice, renaming a remote branch can be done by deleting the old one and then pushing up the new one from your local repository:

```
# First, delete the current / old branch:
```

If you're using a Git desktop GUI like Tower, you won't be bothered with these details: you can simply rename both local and remote branches from a contextual menu (no need to delete and re-push anything):



How to Switch Branches in Git

The current branch (also referred to as the HEAD branch) defines the context you're working in at the moment. Or in other words: the current HEAD branch is where new commits will be created.

Having said that, it makes sense that *switching* the currently active branch is one of the most-used actions any developer performs

branches, you won't be surprised to learn the command that's used to make this happen:

```
$ git checkout <other-branch>
```

However, because the `git checkout` command has so many different duties, the Git community (fairly recently) introduced a new command you can now also use to change the current HEAD branch:

```
$ git switch <other-branch>
```

I think it makes a lot of sense to move away from the `checkout` command – because it's used to perform so many different actions – and instead move towards the new `switch` command, which is absolutely unambiguous about what it does.

How to Publish a Branch in Git

As I already said in the section about "creating branches" above, it's not possible to *create* a new branch on a remote repository.

What we can do, however, is *publish an existing local branch* on a remote repository. We can "upload" what we have locally to the remote and thereby share it with our team:

```
$ git push -u origin <local-branch>
```

one parameter, the `-u` flag, is worth explaining – which I'll do in the next section.

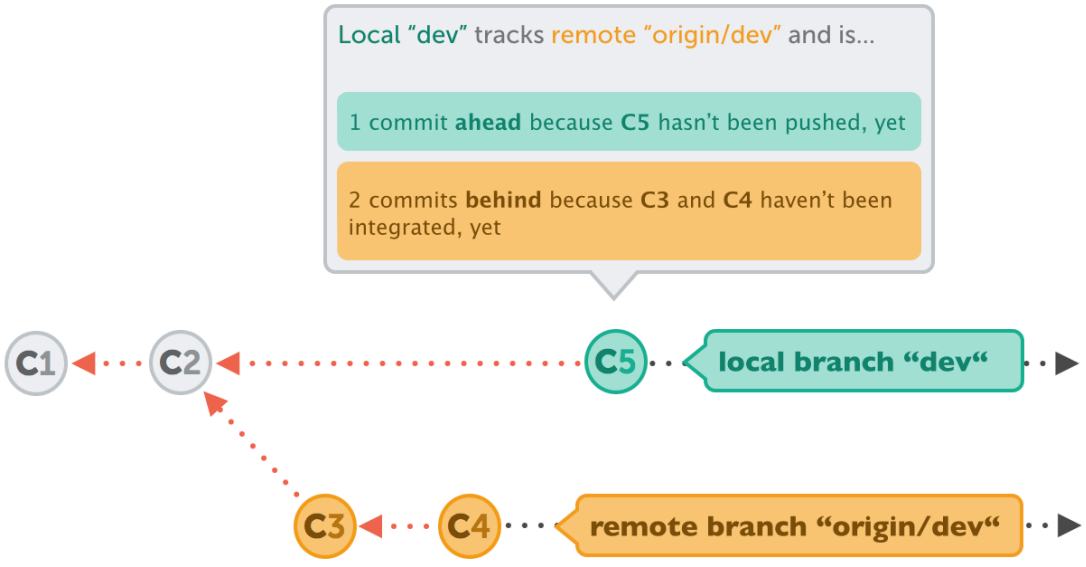
But to give you the short version here: it tells Git to establish a "tracking connection" which will make pushing and pulling much easier in the future.

How to Track Branches in Git

By default, local and remote branches have nothing to do with each other. They are stored and managed as independent objects in Git.

But in real life, of course, local and remote branches often *do* have a relationship with each other. For example, a remote branch is often something like the "counterpart" of a local one.

Such a relationship can be modeled in Git: one branch (typically a local one) can "track" another one (typically remote).



pulling, you can simply use the vanilla commands without any further parameters (for example, a simple `git push`).

The tracking connection helps Git fill in the blanks – which branch on which remote you want to push to, for example.

You have already read about one way to establish such a tracking connection: using `git push` with the `-u` option when publishing a local branch for the first time does exactly that. After that, you can simply use `git push` without mentioning the remote or the target branch.

This also works the other way around: when creating a local branch that should be based on a remote one. In other words, when you want to *track* a remote branch:

```
$ git branch --track <new-branch> origin/<base-branch>
```

Alternatively, you could also use the `git checkout` command to achieve this. If you want to name the local branch after the remote one, you only have to specify the remote branch's name:

```
$ git checkout --track origin/<base-branch>
```

If you want to learn more about this topic, check out this post about "Tracking Relationships in Git".

do a little housecleaning, here's how to delete a local branch:

```
$ git branch -d <branch-name>
```

Note that you might also need the `-f` option in case you're trying to delete a branch that contains un-merged changes. Use this option with care because it makes losing data very easy!

To delete a remote branch, we cannot use the `git branch` command. Instead, `git push` will do the trick, using the `--delete` flag:

```
$ git push origin --delete <branch-name>
```

When deleting a branch, keep in mind that you need to check if you should delete its counterpart branch, too.

For example, if you have just deleted a remote feature branch, it might make sense to also delete its local tracking branch. That way, you make sure you aren't left with lots of obsolete branches – and a messy Git repository.

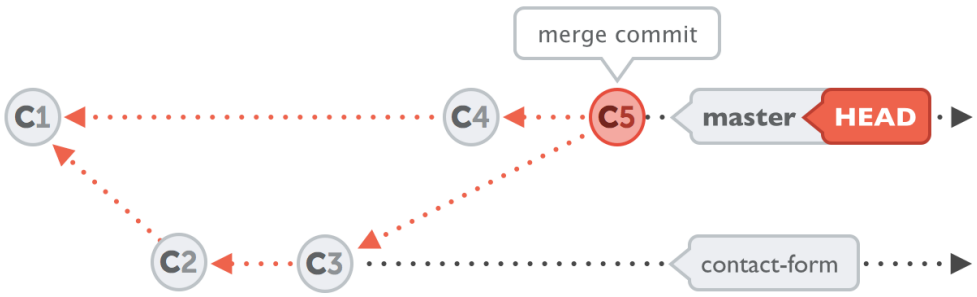
How to Merge Branches in Git

Merging is probably the most popular way to integrate changes. It allows you to bring all of the new commits from another branch into your current HEAD branch.

```
# (1) Check out the branch that should receive the changes
$ git switch main

# (2) Execute the "merge" command with the name of the branch that con
$ git merge feature/contact-form
```

Often, the result of a merge will be a separate new commit, the so-called "merge commit". This is where Git combines the incoming changes. You can think of it like a knot that connects two branches.



There is, of course, a lot more to say about `git merge`. Here are some free resources that help you learn more:

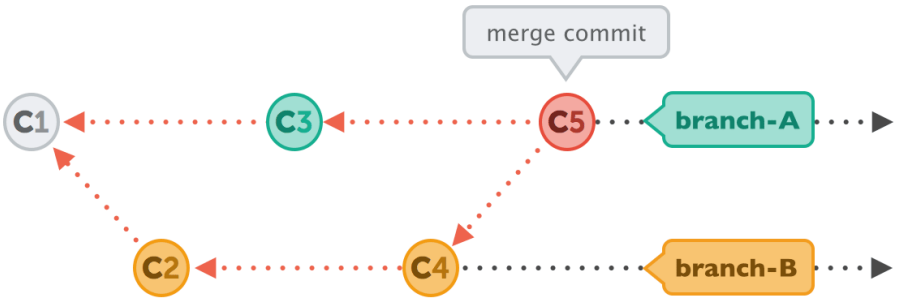
- How to Undo a Merge in Git
- How to Fix and Solve Merge Conflicts
- An Overview of "git merge"

How to Rebase Branches in Git

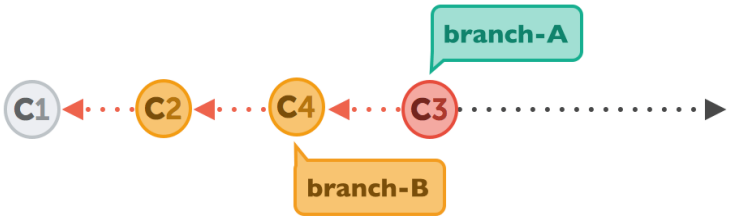
An alternative way to integrate commits from another branch is using `rebase`. And I'm very careful to call it an "alternative" way: it's

preference and the conventions in your team. Some teams love rebase, some prefer merge.

To illustrate the differences between merge and rebase, take a look at the following illustrations. Using `git merge`, the result of our integration of *branch-B* into *branch-A* would look like this:



Using `git rebase`, on the other hand, the end result will look quite different – especially because no separate merge commit will be created. Using rebase, it appears as if your development history happened in a straight line:



Starting the actual process is pretty simple:

```
# (1) Check out the branch that should receive the changes
```

For a deeper understanding of rebase, I recommend the post "Using git rebase instead of git merge".

How to Compare Branches in Git

In certain situations, it can be very helpful to compare two branches. For example, before you decide to integrate or delete a branch, it's interesting to see how it differs from another branch. Does it contain any new commits? And if so: are they valuable?

To see which commits are in branch-B but not in branch-A, you can use the `git log` command with the double dot syntax:

```
$ git log branch-A..branch-B
```

Of course, you could also use this to compare your local and remote states by writing something like `git log main..origin/main`.

If instead of the *commits* you'd prefer to see the *actual changes* that make up those differences, you can use the `git diff` command:

```
$ git diff branch-A..branch-B
```

How to Become More

Git in general: there's a ton of powerful features that many developers don't know or can't use productively.

From Interactive Rebase to Submodules and from the Reflog to File History: it pays to learn these advanced features – by becoming more productive and making fewer mistakes.

One particularly helpful topic is learning how to **undo mistakes with Git**. If you want to dive deeper into how you can save your neck from the inevitable mistakes, check out this video about undoing mistakes in Git.

How to check your Git configuration:?
The command below returns a list of information about your git configuration including user name and email:

```
git config -l
```

How to setup your Git username:
With the command below you can configure your user name:

```
git config --global user.name "Yourname"
```

How to setup your Git user email:
This command lets you setup the user email address you'll use in your commits.

```
git config --global user.email "youremailaddresshere"
```

How to cache your login credentials in Git:
You can store login credentials in the cache so you don't have to type them in each time.
Just use this command:

```
git config --global credential.helper cache
```

How to initialize a Git repo:
Everything starts from here. The first step is to initialize a new Git repo locally in your project root. You can do so with the command below:

```
git init
```

Git & GitHub terminology

Working Directory

It is the folder/directory where we initialize our git repository by using the command

Staging Area

It is an intermediate place between Working Directory and Local Repository to figure out what the things you want git to ignore and what the things you want it to be tracked.

Repository

A repository is like a folder for your project. Your project's repository contains all of your project's files and stores each file's revision history. You can also discuss and manage your project's work within the repository.

Local Repository

It is the repo on which we will make local changes, typically this local repository is on our computer.

Remote Repository

It is a common repository that all team members use to exchange their changes. In most cases, such a remote repository is stored on a code hosting service like GitHub or on an internal server.

Cloning

Cloning a git repository means that you create a local copy of the code provided by developer; it is downloading the whole code of the repository.

Issues

Issues are a great way to keep track of tasks, enhancements, and bugs for your projects. They're kind of like email—except they can be shared and discussed with the rest of your team.

Branches

A branch is a parallel version of a repository. It is contained within the repository, but does not affect the primary or master branch allowing you to work freely without disrupting the "live" version. When you've made the changes you want to make, you can merge your branch back into the master branch to publish your changes.

Master Branch

One word: the master branch is deployable. It is your production code, ready to roll out into the world. The master branch is meant to be stable, and it is the social contract of open source software to never, ever push anything to master that is not tested, or that breaks the build.

Commit

It is a command used to save your changes to the local repository.

Push

Pushing refers to sending your committed changes to a remote repository, such as a repository hosted on GitHub. For instance, if you change something locally, you'd want to then push those changes so that others may access them.

Pull Request

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

Merge

It is a command which lets you take the independent lines of development created by git branch and integrate them into a single branch.

COMMAND SUMMARY:-

Single Git Commands : Initialize a git repo in the current directory

```
git init add --username
git init add --email
```

Add all untracked changed files to staging, ready to be committed

```
git add -A
```

Commit with a message

```
git commit -m "some message here"
```

Review your commit history or checkpoints in time log, (gives you the commit address for reference):

```
git log --oneline
```

Switch to a a previous commit to review code from that checkpoint
git checkout <your-commit address-here>

Sync a remote repository to local repository with automatic alias assigned called "origin"
git remote add <alias-name> <your remote repository link comes here>

Push from local branch "master" up to remote branch "origin" (where 'origin' is set as an alias to the remote repository)
git push origin master

Pull from remote branch "origin" down to local branch "master"
git pull origin master

Create a new branch to manage a side hustle in project
git branch <your new branch name>

View all branches in your repository, also marks the branch you are working on with *
git branch -a

Switch to a branch to work in it
git checkout <your-branch name-here>

Delete a branch
git checkout master
// Deleting local branch
git branch -D <branch name to be deleted>

How to add a file to the staging area in Git:
The command below will add a file to the staging area. Just replace filename_here with the name of the file you want to add to the staging area.

```
git add filename_here
```

How to add all files in the staging area in Git
If you want to add all files in your project to the staging area, you can use a wildcard . and every file will be added for you.

```
git add .
```

How to add only certain files to the staging area in Git
With the asterisk in the command below, you can add all files starting with 'fil' in the staging area.

```
git add fil*
```

How to check a repository's status in Git:
This command will show the status of the current repository including staged, unstaged, and untracked files.

```
git status
```

How to commit changes with a message in Git:
You can add a commit message without opening the editor. This command lets you only specify a short summary for your commit message.

```
git commit -m "your commit message here"
```

How to commit changes (and skip the staging area) in Git:
You can add and commit tracked files with a single command by using the -a and -m options.
```
git commit -a -m"your commit message here"
```

How to see your commit history in Git:
This command shows the commit history for the current repository:
```
git log
```

How to see your commit history including changes in Git:
This command shows the commit's history including all files and their changes:
```
git log -p
```

How to see a specific commit in Git:
This command shows a specific commit.
Replace commit-id with the id of the commit that you find in the commit log after the word commit.
```
git show commit-id
```

How to see log stats in Git:
This command will cause the Git log to show some statistics about the changes in each commit, including line(s) changed and file names.
```
git log --stat
```

How to see changes made before committing them using "diff" in Git:
You can pass a file as a parameter to only see changes on a specific file. git diff shows only unstaged changes by default.

We can call diff with the --staged flag to see any staged changes.
git diff
git diff all_checks.py
git diff --staged

How to see changes using "git add -p":
This command opens a prompt and asks if you want to stage changes or not, and includes other options.
git add -p

How to remove tracked files from the current working tree in Git:
This command expects a commit message to explain why the file was deleted.
git rm filename

How to rename files in Git:
This command stages the changes, then it expects a commit message.
git mv oldfile newfile

How to ignore files in Git:
Create a .gitignore file and commit it.

How to revert unstaged changes in Git:
git checkout filename

How to revert staged changes in Git:
You can use the -p option flag to specify the changes you want to reset.
git reset HEAD filename
git reset HEAD -p

How to amend the most recent commit in Git:
allows you to modify and add changes to the most recent commit.
!!Note!!: fixing up a local commit with amend is great and you can push it to a shared repository after you've fixed it. But you should avoid amending commits that have already been made public.
git commit --amend

How to rollback the last commit in Git:
git revert will create a new commit that is the opposite of everything in the given commit.
We can revert the latest commit by using the head alias like this:
git revert HEAD

How to rollback an old commit in Git:
You can revert an old commit using its commit id. This opens the editor so you can add a commit message.
git revert comit_id_here

How to create a new branch in Git:
By default, you have one branch, the main branch. With this command, you can create a new branch. Git won't switch to it automatically – you will need to do it manually with the next command.
git branch branch_name

How to switch to a newly created branch in Git:
When you want to use a different or a newly created branch you can use this command:
`git checkout branch_name`

How to list branches in Git:
You can view all created branches using the git branch command. It will show a list of all branches and mark the current branch with an asterisk and highlight it in green.
`git branch`

How to create a branch in Git and switch to it immediately:
In a single command, you can create and switch to a new branch right away.
`git checkout -b branch_name`

How to delete a branch in Git:
When you are done working with a branch and have merged it, you can delete it using the command below:
`git branch -d branch_name`

How to merge two branches in Git:
To merge the history of the branch you are currently in with the branch_name, you will need to use the command below:
`git merge branch_name`

How to show the commit log as a graph in Git:
We can use --graph to get the commit log to show as a graph. Also, --oneline will limit commit messages to a single line.
git log --graph --oneline

How to show the commit log as a graph of all branches in Git:
Does the same as the command above, but for all branches.
git log --graph --oneline --all

How to abort a conflicting merge in Git:
If you want to throw a merge away and start over, you can run the following command:
git merge --abort

How to add a remote repository in Git
This command adds a remote repository to your local repository
(just replace https://repo_here with your remote repo URL).
git add remote https://repo_here

How to see remote URLs in Git:
You can see all remote repositories for your local repository with this command:
git remote -v

How to get more info about a remote repo in Git:
Just replace origin with the name of the remote obtained by
running the git remote -v command.
git remote show origin

How to push changes to a remote repo in Git:
When all your work is ready to be saved on a remote repository, you can push all changes using the command below:

```
git push
```

How to pull changes from a remote repo in Git:
If other team members are working on your repository, you can retrieve the latest changes made to the remote repository with the command below:

```
git pull
```

How to check remote branches that Git is tracking:
This command shows the name of all remote branches that Git is tracking for the current repository:

```
git branch -r
```

How to fetch remote repo changes in Git:
This command will download the changes from a remote repo but will not perform a merge on your local branch (as git pull does that instead).

```
git fetch
```

How to check the current commits log of a remote repo in Git
Commit after commit, Git builds up a log. You can find out the remote repository log by using this command:

```
git log origin/main
```

How to merge a remote repo with your local repo in Git:
If the remote repository has changes you want to merge with your local,
then this command will do that for you:
git merge origin/main

How to get the contents of remote branches in Git without automatically merging:
This lets you update the remote without merging any content into the
local branches. You can call git merge or git checkout to do the merge.
git remote update

How to push a new branch to a remote repo in Git:
If you want to push a branch to a remote repository you can use the command below.
Just remember to add -u to create the branch upstream:
git push -u origin branch_name

How to remove a remote branch in Git:
If you no longer need a remote branch you can remove it using the command below:
git push --delete origin branch_name_here

How to use Git rebase:
You can transfer completed work from one branch to another using git rebase.
git rebase branch_name_here

How to run rebase interactively in Git:
You can run git rebase interactively using the -i flag.
It will open the editor and present a set of commands you can use.

```
git rebase -i master
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
```

How to force a push request in Git:
This command will force a push request. This is usually fine for pull request branches because nobody else should have cloned them. But this isn't something that you want to do with public repos.

```
git push -f
```