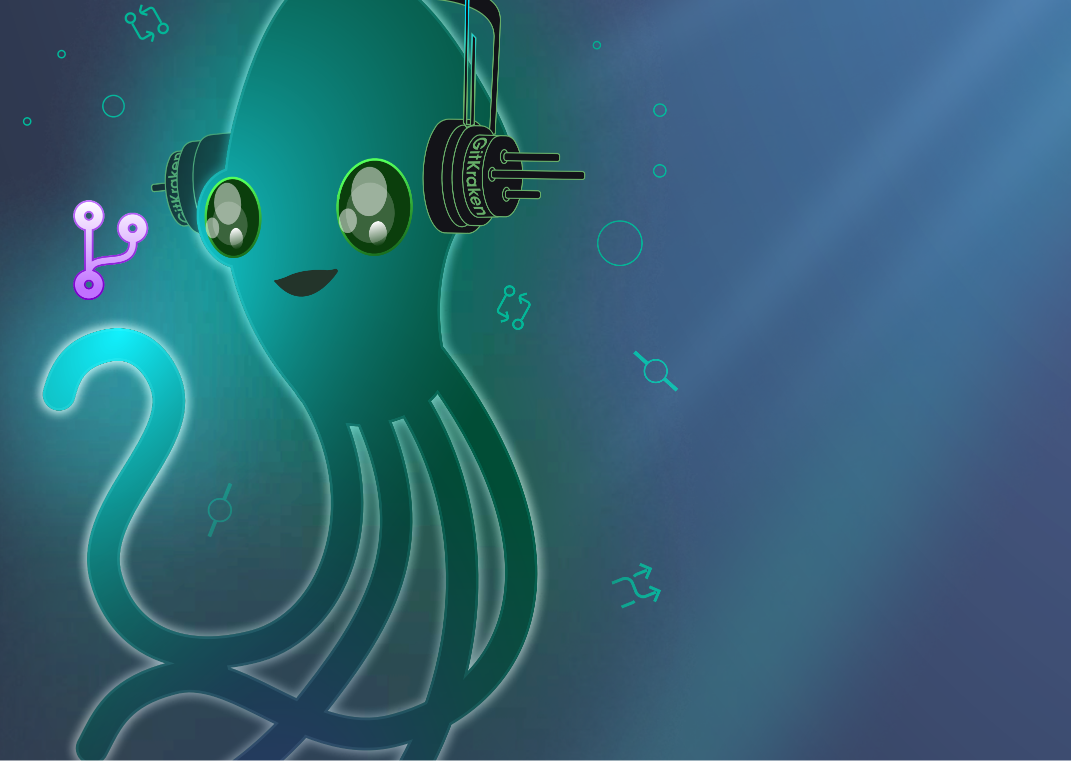
CHEAT SHEET

THE BASICS OF GIT



Git Config

Set your user information to be used in the commit history.

```
$ git config --global user.name "[name]"
```

Sets the name you want to attach to your commit transactions.

```
$ git config --global user.email "[email address]"
```

Sets the email you want to attach to your commit transactions.

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

Enables helpful colorization of the command line output.

Creating and Cloning Repositories

You can create a new git repository locally, or make a copy of an existing online repository to work from.

```
$ git init
```

The git init command turns the directory where you ran the command into a new Git repository by adding a .git folder and instructs Git to start watching for changes.

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

The Clone command downloads a copy of a repository from a remote server, or a service like GitHub, Gitlab, or Bitbucket. This includes all of the files and commits, as well as connections to all the remote branches.

Making Changes

Git makes it easy to track changes to files over time. Once you have initialized or cloned a repository, all of your changes are in a Work In Process state and Git can see what changes have been made. However, in order for Git to track a snapshot of your changes, you will need to add and commit your work.

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

Stages a single file in preparation for committing.

```
$ git add .
```

Stages all the changed files in your current directory in preparation for committing.

Making Changes Cont'd

```
$ git commit -m "[descriptive  message]"
```

Makes a permanent record of the file snapshots currently staged in your versioning history and includes a short message about what you changed.

Note: If you run `git commit` without `-m` , you will find yourself in the default text editor of your terminal, which is very commonly `vim` . From here you can write your commit message, save and exit the editor to complete the commit.

Viewing Changes

Git offers a lot of visibility to the state of your current work, your history of commits, and fine-grain details about those changes.

```
$ git status
```

Shows you information about what state Git is in, including which branch you are on and if you are up to date with the remote repositories.

```
$ git log
```

Lists version history for the current branch, including full commit message, author, time of the commit, and the commit hash.

```
$ git log --oneline
```

Lists a shortened version history for the current branch, only showing the first line of the commit message and commit hash.

```
$ git diff
```

Shows all changes across files that have not been staged yet.

```
$ git show [commit]
```

Shows the metadata and content changes of the specified commit. The commit is identified by the hash, which is a unique id number generated when commits are made.

The .gitignore file

There are certain files that you might not want to track, such as images and videos or files that contain sensitive information like credentials. You can do this by creating a special file named `.gitignore.` Inside this file you can list individual files to ignore or use the * wildcard to exclude entire types of files, such as *.jpg or *.mp4.

Ready to simplify complex Git commands? Learn Git quickly & increase productivity with the GitKraken Git GUI!

Download GitKraken Desktop For Free

GITKRAKEN CHEAT SHEET : THE BASICS OF GIT

Undoing Changes and Commits

There are two main ways to make changes to your commit history.

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

Creates a new commit that undoes the changes introduced by the specified commit.

This is a safe way to reverse changes in a shared history because it preserves commit history and doesn't rewrite it.

```
$ git reset [commit]
```

Undoes all commits after `[commit]` and preserves changes locally. While powerful, this carries a high risk of merge conflicts if you have made a lot of changes since that commit.

```
$ git reset --hard [commit]
```

Moves the current branch to the specified commit and updates the working directory and index to match.

⚠ Warning: This permanently deletes changes and rewrites history. Avoid using on shared branches unless you're sure others haven't based work on the affected commits.

Connecting to Remote Repositories

Local repositories can have connections to one or more remote repositories in order to push changes to them or pull changes from them. If you clone a repository, the remote URL you cloned it from will automatically be set as `origin` as the remote name.

```
$ git remote add [remote name] [url]
```

Specifies the remote repository for your local repository. The `[url]` points to a repository on a remote server or service like GitHub, Gitlab, or Bitbucket.

Synchronize Changes

Synchronize your local repository with the remote repository.

```
$ git push [remote name] [branch]
```

Uploads all local commits from your local history to your specified remote and named branch.

```
$ git fetch
```

Downloads all history from the remote branch but does not automatically commit them.

```
$ git merge
```

Combines the fetched history from the remote branch into the current local branch.

```
$ git pull
```

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

⚠ Tip: Git merges can be tricky. You can use GitKraken AI to make them easier! Learn About GitKraken AI

Branches

Branches allow you to make commits that do not affect your other work until you are ready to apply those changes. Any commits will be made on the branch you have currently "checked out" and not the default 'main' or 'master' branch. You can always use `git status` to see which branch you are on.

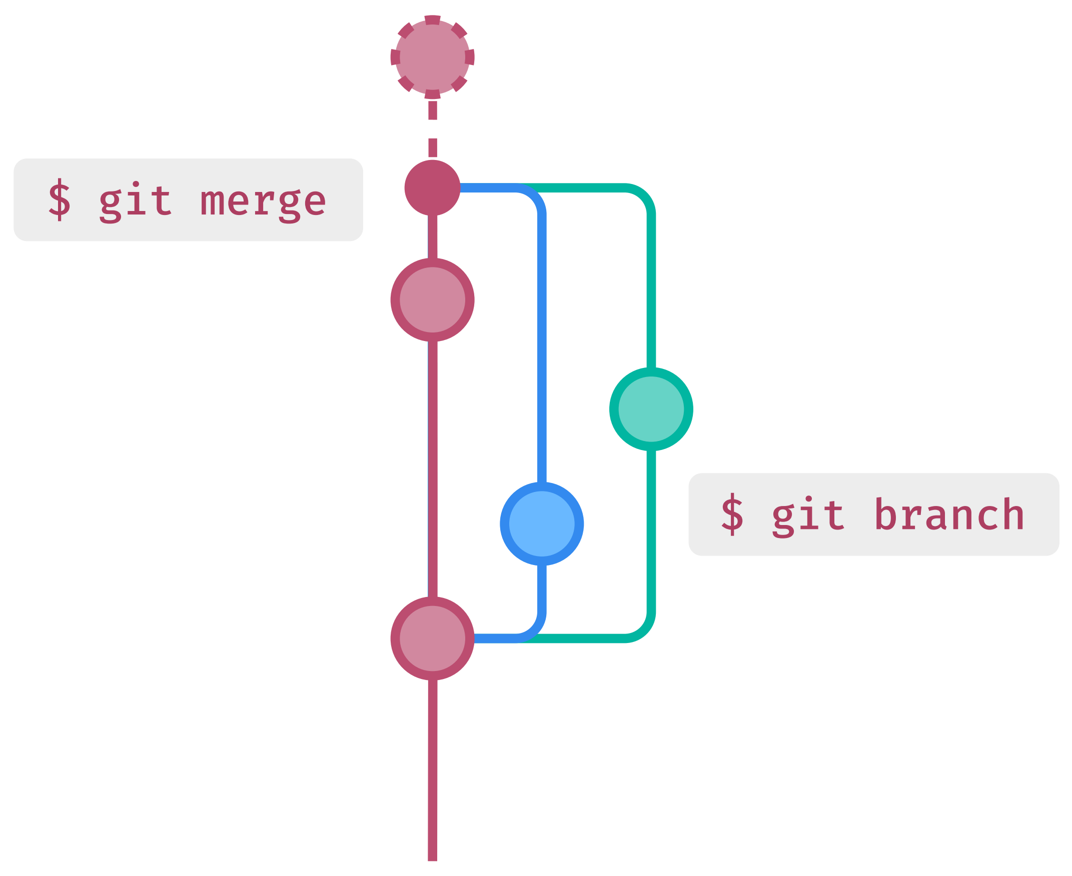
```
$ git branch [branch-name]
```

Creates a new branch. For example: `git branch feature-A`

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

Switches to the specified branch and updates any documents to use that version.

Git Branching diagram:



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

Combines the specified branch's history into the current branch. Always think of Git pulling things into it. In this case, pulling the changes from the specified `[branch-name]` into your currently checked out branch.

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

Deletes the specified branch

Learn more Git commands & concepts online! Visit Our Learning Center →