1. Used Most Frequently

```
# 1 - Start a working area
$ git init          # Create an empty Git repository or reinitialize an existing one
$ git clone         # Clone a repository into a new directory

# Work on the current change
$ git add           # Add file contents to the index
$ git mv            # Move or rename a file, a directory, or a symlink
$ git restore       # Restore working tree files
$ git rm            # Remove files from the working tree and from the index

# 2 - Examine the history and state

$ git bisect        # Use binary search to find the commit that introduced a bug
$ git diff          # Show changes between commits, commit and working tree, etc
$ git grep          # Print lines matching a pattern
$ git log           # Show commit logs
$ git show          # Show various types of objects
$ git status        # Show the working tree status

# 3 - Grow, mark and tweak your common history
$ git branch        # List, create, or delete branches
$ git commit        # Record changes to the repository
$ git merge         # Join two or more development histories together
$ git rebase        # Reapply commits on top of another base tip
$ git reset         # Reset current HEAD to the specified state
$ git switch        # Switch branches
$ git tag            # Create, list, delete or verify a tag object signed with GPG

# 4 - Collaborate
$ git fetch     # Download objects and refs from another repository
$ git pull      # Fetch from and integrate with another repository or a local branch
$ git push      # Update remote refs along with associated objects
```

2. Basic Concepts

Three states of a file

- **modified**

  You have changed the file but have not committed it to your database yet.

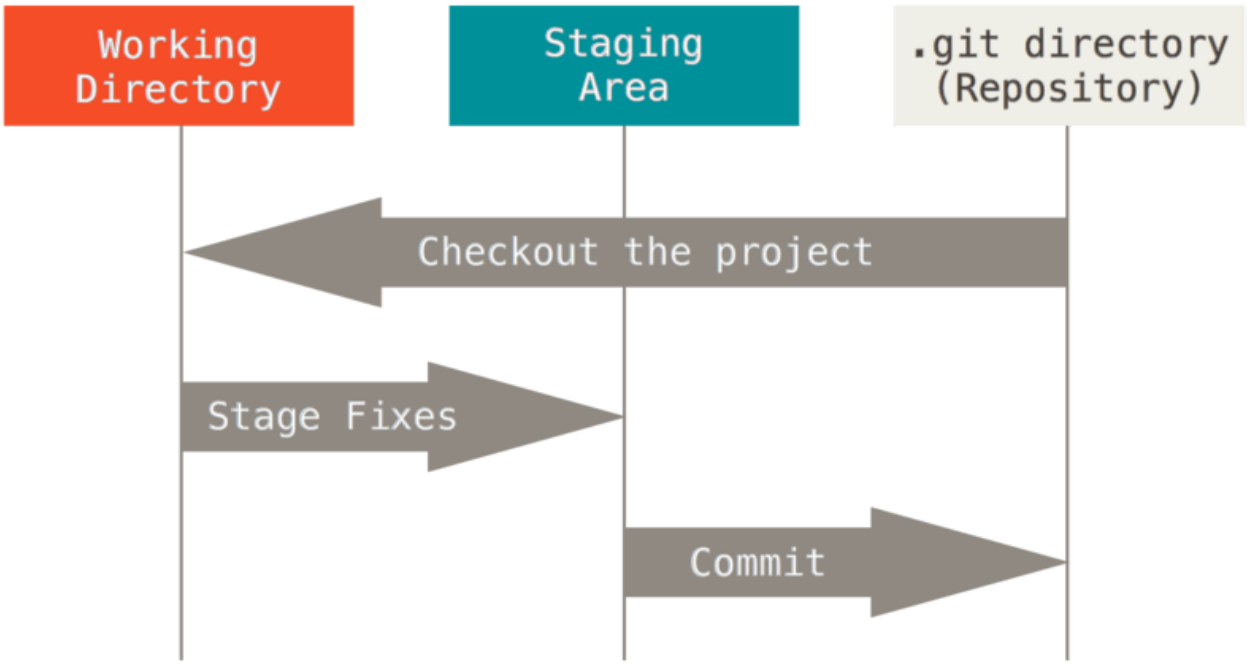
- **staged**

  You have marked a modified file in its current version to go into your next commit snapshot.

- **committed**

  The data is safely stored in your local database.

Three Sections of a Git project



Basic Git workflow

1. You modify files in your working tree.
2. You selectively stage just those changes you want to be part of your next commit, which adds only those changes to the staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

3. Git Cheat Sheet

Configration

```
# Global Git configration : ~/.gitconfig
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "your_email@example.com"

$ git config --global color.ui auto
$ git config --global color.diff auto
$ git config --global color.branch auto

$ git config --global core.editor emacs
$ git config --global init.defaultBranch main

$ git config --list
$ git config user.name
```

Create

```
# Initializing a Repository in an Existing Directory
$ cd /Users/user/my_project
$ git init

# Cloning an Existing Repository
$ git clone https://github.com/libgit2/libgit2 mylibgit

# Tracking New Files
$ git add file_name
```

Browse

```
$ git status
$ git log
$ git log --graph
$ git diff
$ git branch
$ git show
```

Revert

```
$ git reset
$ git checkout
$ git revert
```

Update

```
$ git pull
$ git merge
$ git fetch
$ git am
```

Branch

```
$ git checkout -b
$ git checkout -b feature-A
$ git checkout master
$ git checkout -
```

- `origin` : default upstream repository
- `HEAD` : current branch
- `HEAD^` : parent of current branch
- `HEAD~4` : great-great grandparent of current branch

Commit and Publish

```
$ git commit
$ git commit --amend

$ git rebase -i
$ git push
```

Do not forget `git --help`.

Reference: Pro Git