Git Cheat Sheet

taraba.

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

01 Git history

Git was created in 2005 by **Linus Torvalds**. The reason for its creation was that another source-control management tool named BitKeeper become a proprietary SCM system. Linus needed an SCM for the development of the Linux kernel and he decided to rather build a new one from scratch then move to some other.

Some of the goals of the new system were: speed, simple design, strong support for non-linear development, fully distributed, ability to handle large projects like the Linux kernel efficiently (speed and data size). Git has evolved and matured to be easy to use and yet retain these initial qualities. It's amazingly fast, it's very efficient with large projects, and it has an incredible branching system for non-linear development

02 Git installation

To check do you already have Git installed:

```
which git
```

Version check:

```
git --version
```

Mac installation

1. Apple may have pre-installed a version of Git
2. Installer
   http://git-scm.com/download/mac
   Download and run the installer program
   .pkg -> if security warning pop up, press control key + open with Installer
3. Homebrew installation
   https://brew.sh
   brew install git

Windows installation

- http://git-scm.com
- Download for Windows | follow instructions

Linux installation

- http://git-scm.com/download/linux

- Find command for your Linux distribution
    - example for Debian/Ubuntu: `apt-get install git`

03 Git configuration

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

```
git config --global user.name "Milomir Dragovic"
```

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

```
git config --global user.email "hello@milomir.rs"
```

Enable some colorization of Git output.

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

04 Starting a project

Create a new local repository. If the project name is not provided repository will be instantiated in the current directory.

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

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

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

05 Daily usage

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 status
```

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 add [file]
```

Show changes between the working directory and staging area.

```
git diff [file]
```

Shows any changes between the staging area and the repository.

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

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

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

Revert your repository to a previously known working state.

```
git reset [file]
```

Create a new commit from changes added to the staging area.

```
git commit -m "explain what this change will do"
```

Remove file from the working directory and staging area.

```
git rm [file]
```

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

```
git stash
```

List stack-order of stashed file changes

```
git stash list
```

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

```
git stash pop
```

Delete a specific stash from all your previous stashes.

```
git stash drop
```

06 Branching

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

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

Create a new branch, referencing the current HEAD.

```
git branch [branch_name]
```

Switches to the specified branch and updates the working directory.

```
 git checkout [branch_name]
```

Git will create the specified branch if it does not exist and switch to it.

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

Merge specified branch into the current one. This is usually done in pull requests.

```
git merge [branch_name]
```

Delete selected branch if it is already merged into any other.

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

Remove selected branch, no matter if it is already merged into any other.

```
git branch -D [name]
```

07 Synchronize changes with remote repositories

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

```
git fetch [remote]
```

Delete remote Refs that were removed from the remote repository.

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

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

```
git pull [remote]
```

Push local changes to the remote.

```
git push [remote]
```

Push local changes with tags to the remote.

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

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

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

08 Going through the history

List commit history of the current branch.

```
git log
```

List last n commits of the current branch.

```
git log [-n]
```

Condense each commit to a single line.

```
git log --oneline
```

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

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

Lists version history for a file, including renames

```
git log --follow [file]
```

Shows content differences between two branches

```
git diff [first-branch]...[second-branch]
```

Outputs metadata and content changes of the specified commit

```
git show [commit SHA]
```

Search for commits by a particular author.

```
git log --author="name"
```

09 Rewriting history

Replace the last commit with the staged changes and the last commit combined. Use with nothing staged to edit the last commit's message.

```
git commit --amend
```

Apply any commits of a current branch ahead of the specified one

```
git rebase [branch]
```

Clear staging area, rewrite working tree from the specified commit.

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

10 Tagging

List all tags.

```
git tag
```

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

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

Create a tag object named name for the current commit.

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

Remove a tag from the local repository.

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

11 Ignoring files

Verify the .gitignore file exists in your project and ignore certain types of files, such as:

- compiled source code
- packages and compressed files
- logs and databases
- operating system generated files
- user-uploaded assets (images, PDFs, videos)

For creating a .gitignore file you can use some of the publicly available generators like this one:

- https://www.toptal.com/developers/gitignore.

More info:

- https://help.github.com/articles/ignoring-files
- https://github.com/github/gitignore