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

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

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

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

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

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

Git Cheat Sheet

GitLab

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 in the current directory. If **[project name]** is provided, Git will create a new directory named **[project name]** and will initialize a repository inside it.
`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 [<path>...]`	Revert some paths in the index (or the whole index) to their state in **HEAD.**
`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.**

04 Storing your work

`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.**

05 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 rebase [branch_name]`	Apply commits of the current working branch and apply them to the HEAD of [branch] to make the history of your branch more linear.
`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 [branch_name]`	Join specified **[branch_name]** branch into your current branch (the one you are on currently).
`git branch -d [branch_name]`	Remove selected branch, if it is already merged into any other. **-D** instead of **-d** forces deletion.

Commit	a state of the code base
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

06 Inspect history

`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.

07 Tagging 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.

08 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. It's easy to lose uncommitted changes with **--hard.**
`git revert [commit sha]`	Create a new commit, reverting changes from the specified commit. It generates an **inversion** of changes.

09 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.

10 Git installation

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

11 Ignoring files

```
cat  <<EOF > .gitignore
/logs/*
!logs/.gitkeep
/tmp
*.swp
EOF
```

To ignore files, create a .gitignore file in your repository with a line for each pattern. File ignoring will work for the current and sub directories where .gitignore file is placed. In this example, all files are ignored in the logs directory (excluding the .gitkeep file), whole tmp directory and all files *.swp.