

Git Cheat Sheet



GIT BASICS

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

UNDOING CHANGES

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

REWRITING GIT HISTORY

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

GIT BRANCHES

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

REMOTE REPOSITORIES

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

Additional Options +

GIT CONFIG

<code>git config --global user.name <name></code>	Define the author name to be used for all commits by the current user.
<code>git config --global user.email <email></code>	Define the author email to be used for all commits by the current user.
<code>git config --global alias. <alias-name> <git-command></code>	Create shortcut for a Git command. E.g. <code>alias.glog "log --graph --oneline"</code> will set "git glog" equivalent to "git log --graph --oneline".
<code>git config --system core.editor <editor></code>	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).
<code>git config --global --edit</code>	Open the global configuration file in a text editor for manual editing.

GIT LOG

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

<code>git diff HEAD</code>	Show difference between working directory and last commit.
<code>git diff --cached</code>	Show difference between staged changes and last commit

GIT RESET

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

GIT REBASE

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

GIT PULL

<code>git pull --rebase <remote></code>	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

<code>git push <remote> --force</code>	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.
<code>git push <remote> --all</code>	Push all of your local branches to the specified remote.
<code>git push <remote> --tags</code>	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.

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/*
```

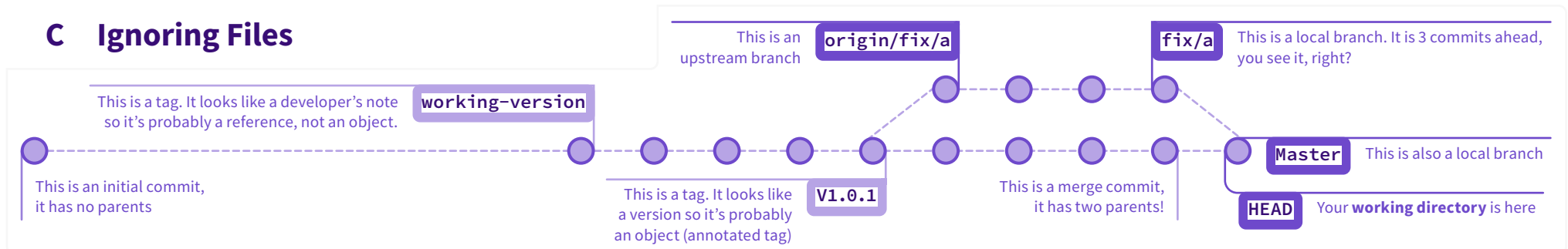
```
!logs/.gitkeep
```

```
/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

