

# Git Cheat Sheet



## Git Basics

<code>git init</code> <code>&lt;directory&gt;</code>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
<code>git clone &lt;repo&gt;</code>	Clone repo located at <code>&lt;repo&gt;</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 &lt;name&gt;</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>&lt;directory&gt;</code>	Stage all changes in <code>&lt;directory&gt;</code> for the next commit. Replace <code>&lt;directory&gt;</code> with a <code>&lt;file&gt;</code> to change a specific file.
<code>git commit -m</code> " <code>&lt;message&gt;</code> "	Commit the staged snapshot, but instead of launching a text editor, use <code>&lt;message&gt;</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>&lt;commit&gt;</code>	Create new commit that undoes all of the changes made in <code>&lt;commit&gt;</code> , then apply it to the current branch.
<code>git reset &lt;file&gt;</code>	Remove <code>&lt;file&gt;</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 &lt;base&gt;</code>	Rebase the current branch onto <code>&lt;base&gt;</code> . <code>&lt;base&gt;</code> can be a commit ID, a 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>&lt;branch&gt;</code> argument to create a new branch with the name <code>&lt;branch&gt;</code> .
<code>git checkout -b</code> <code>&lt;branch&gt;</code>	Create and check out a new branch named <code>&lt;branch&gt;</code> . Drop the <code>-b</code> flag to checkout an existing branch.
<code>git merge &lt;branch&gt;</code>	Merge <code>&lt;branch&gt;</code> into the current branch.

## Remote Repositories

<code>git remote add</code> <code>&lt;name&gt; &lt;url&gt;</code>	Create a new connection to a remote repo. After adding a remote, you can use <code>&lt;name&gt;</code> as a shortcut for <code>&lt;url&gt;</code> in other commands.
<code>git fetch</code> <code>&lt;remote&gt; &lt;branch&gt;</code>	Fetches a specific <code>&lt;branch&gt;</code> , from the repo. Leave off <code>&lt;branch&gt;</code> to fetch all remote refs.
<code>git pull &lt;remote&gt;</code>	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
<code>git push</code> <code>&lt;remote&gt; &lt;branch&gt;</code>	Push the branch to <code>&lt;remote&gt;</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 &lt;name&gt;</code>	Define the author name to be used for all commits by the current user.
<code>git config --global user.email &lt;email&gt;</code>	Define the author email to be used for all commits by the current user.
<code>git config --global alias. &lt;alias-name&gt; &lt;git-command&gt;</code>	Create shortcut for a Git command. E.g. <code>alias.glog log --graph --oneline</code> will set <code>git glog</code> equivalent to <code>git log --graph --oneline</code> .
<code>git config --system core.editor &lt;editor&gt;</code>	Set text editor used by commands for all users on the machine. <code>&lt;editor&gt;</code> 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 -&lt;limit&gt;</code>	Limit number of commits by <code>&lt;limit&gt;</code> . E.g. <code>git log -5</code> 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="&lt;pattern&gt;"</code>	Search for commits by a particular author.
<code>git log --grep="&lt;pattern&gt;"</code>	Search for commits with a commit message that matches <code>&lt;pattern&gt;</code> .
<code>git log &lt;since&gt;..&lt;until&gt;</code>	Show commits that occur between <code>&lt;since&gt;</code> and <code>&lt;until&gt;</code> . Args can be a commit ID, branch name, HEAD, or any other kind of revision reference.
<code>git log -- &lt;file&gt;</code>	Only display commits that have the specified file.
<code>git log --graph --decorate</code>	<code>--graph</code> flag draws a text based graph of commits on left side of commit msgs. <code>--decorate</code> 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 <b>overwrites all changes</b> in the working directory.
<code>git reset &lt;commit&gt;</code>	Move the current branch tip backward to <code>&lt;commit&gt;</code> , reset the staging area to match, but leave the working directory alone.
<code>git reset --hard &lt;commit&gt;</code>	Same as previous, but resets both the staging area & working directory to match. <b>Deletes uncommitted changes, and all commits after &lt;commit&gt;</b> .

### git rebase

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

### git pull

<code>git pull --rebase &lt;remote&gt;</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 &lt;remote&gt; --force</code>	Forces the <code>git push</code> even if it results in a non-fast-forward merge. Do not use the <code>--force</code> flag unless you're absolutely sure you know what you're doing.
<code>git push &lt;remote&gt; --all</code>	Push all of your local branches to the specified remote.
<code>git push &lt;remote&gt; --tags</code>	Tags aren't automatically pushed when you push a branch or use the <code>--all</code> flag. The <code>--tags</code> flag sends all of your local tags to the remote repo.

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 committing 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

# GitHub Education

Teach and learn better, together. GitHub is free for students and teachers. Discounts available for other educational uses.

✉ [education@github.com](mailto:education@github.com)  
🌐 [education.github.com](https://education.github.com)

# Git Cheat Sheet

by Jan Krüger <jk@jk.gs>, <http://jan-krueger.net/git/>  
Based on work by Zack Rusin

## Basics

Use git help [command] if you're stuck.

master	default devel branch
origin	default upstream branch
HEAD	current branch
HEAD^	parent of HEAD
HEAD~4	great-great grandparent of HEAD
foo..bar	from branch foo to branch bar

## Create

### From existing files

```
git init
git add .
```

### From existing repository

```
git clone ~/old ~/new
git clone git://...
git clone ssh://...
```

## View

```
git status
git diff [oldid newid]
git log [-p] [file|dir]
git blame file
git show id (meta data + diff)
git show id:file
git branch (shows list, * = current)
git tag -l (shows list)
```

## Revert

In Git, revert usually describes a new commit that undoes previous commits.

```
git reset --hard (NO UNDO)
    (reset to last commit)
git revert branch
git commit -a --amend
    (replaces prev. commit)
git checkout id file
```

## Publish

In Git, commit only respects changes that have been marked explicitly with add.

```
git commit [-a]
    (-a: add changed files
    automatically)
git format-patch origin
    (create set of diffs)
git push remote
    (push to origin or remote)
git tag foo
    (mark current version)
```

## Update

```
git fetch (from def. upstream)
git fetch remote
git pull (= fetch & merge)
git am -3 patch.mbox
git apply patch.diff
```

## Branch

```
git checkout branch
    (switch working dir to branch)
git merge branch
    (merge into current)
git branch branch
    (branch current)
git checkout -b new other
    (branch new from other and
    switch to it)
```

## Useful Tools

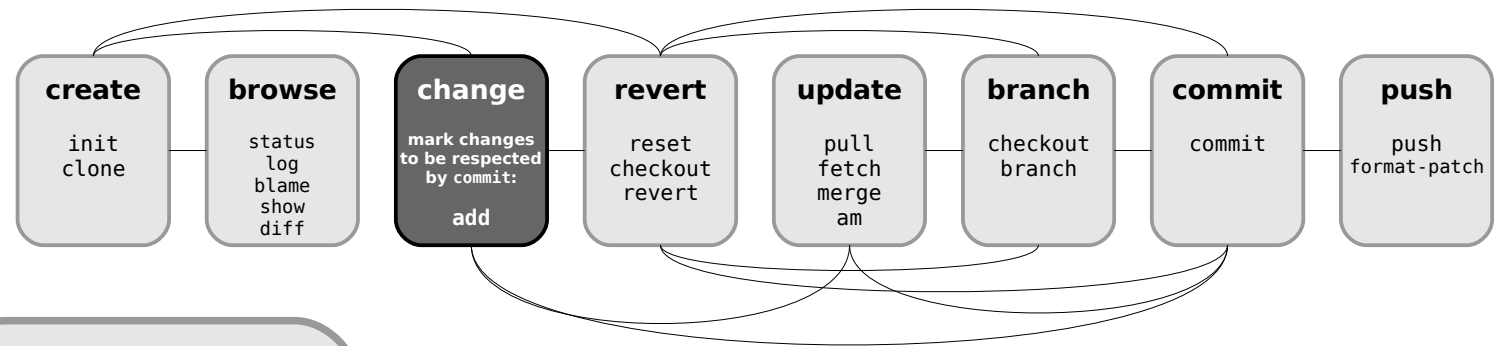
```
git archive
    Create release tarball
git bisect
    Binary search for defects
git cherry-pick
    Take single commit from elsewhere
git fsck
    Check tree
git gc
    Compress metadata (performance)
git rebase
    Forward-port local changes to
    remote branch
git remote add URL
    Register a new remote repository
    for this tree
git stash
    Temporarily set aside changes
git tag
    (there's more to it)
gitk
    Tk GUI for Git
```

## Conflicts

Use add to mark files as resolved.

```
git diff [--base]
git diff --ours
git diff --theirs
git log --merge
gitk --merge
```

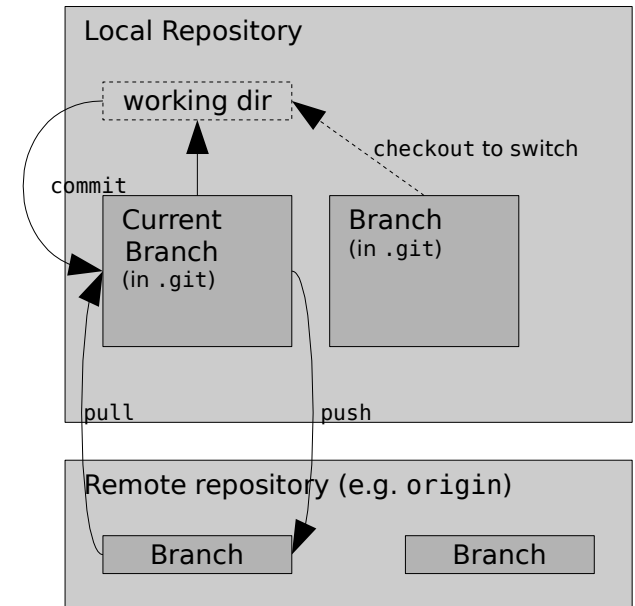
## (left to right) Command Flow



## Tracking Files

```
git add files
git mv old new
git rm files
git rm --cached files
    (stop tracking but keep files in working dir)
```

## Structure Overview



# git cheat sheet

learn more about git the simple way at [rogerdudler.github.com/git-guide/](https://rogerdudler.github.com/git-guide/)  
cheat sheet created by Nina Jaeschke of [ninagrafik.com](https://ninagrafik.com)

## create & clone

<b>create new repository</b>	<code>git init</code>
<b>clone local repository</b>	<code>git clone /path/to/repository</code>
<b>clone remote repository</b>	<code>git clone username@host:/path/to/repository</code>

## add & remove

<b>add changes to INDEX</b>	<code>git add &lt;filename&gt;</code>
<b>add all changes to INDEX</b>	<code>git add *</code>
<b>remove/delete</b>	<code>git rm &lt;filename&gt;</code>

## commit & synchronize

<b>commit changes</b>	<code>git commit -m "Commit message"</code>
<b>push changes to remote repository</b>	<code>git push origin master</code>
<b>connect local repository to remote repository</b>	<code>git remote add origin &lt;server&gt;</code>
<b>update local repository with remote changes</b>	<code>git pull</code>

## branches

<b>create new branch</b>	<code>git checkout -b &lt;branch&gt;</code> <small>e.g. <code>git checkout -b feature_x</code></small>
<b>switch to master branch</b>	<code>git checkout master</code>
<b>delete branch</b>	<code>git branch -d &lt;branch&gt;</code>
<b>push branch to remote repository</b>	<code>git push origin &lt;branch&gt;</code>

## merge

<b>merge changes from another branch</b>	<code>git merge &lt;branch&gt;</code>
<b>view changes between two branches</b>	<code>git diff &lt;source_branch&gt; &lt;target_branch&gt;</code> <small>e.g. <code>git diff feature_x feature_y</code></small>

## tagging

<b>create tag</b>	<code>git tag &lt;tag&gt; &lt;commit ID&gt;</code> <small>e.g. <code>git tag 1.0.0 1b2e1d63ff</code></small>
<b>get commit IDs</b>	<code>git log</code>

## restore

<b>replace working copy with latest from HEAD</b>	<code>git checkout -- &lt;filename&gt;</code>
---	---

### Tip

Want a simple but powerful  
git-client for your mac?

Try Tower: [www.git-tower.com/](https://www.git-tower.com/)



# Git Cheat Sheet

<http://git.or.cz/>

Remember: `git command --help`

Global Git configuration is stored in `$HOME/.gitconfig` (`git config --help`)

## Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

## Show

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

What changed between \$ID1 and \$ID2

```
git diff $id1 $id2
```

History of changes

```
git log
```

History of changes for file with diffs

```
git log -p $file $dir/ec/tory/
```

Who changed what and when in a file

```
git blame $file
```

A commit identified by \$ID

```
git show $id
```

A specific file from a specific \$ID

```
git show $id:$file
```

All local branches

```
git branch
```

(star \* marks the current branch)

## Concepts

### Git Basics

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

### Revert

Return to the last committed state

```
git reset --hard
```



you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commit

```
git revert $id
```

Creates a new commit

Fix the last commit

```
git commit -a --amend
```

(after editing the broken files)

Checkout the \$id version of a file

```
git checkout $id $file
```

### Branch

Switch to the \$id branch

```
git checkout $id
```

Merge branch1 into branch2

```
git checkout $branch2
git merge branch1
```

Create branch named \$branch based on the HEAD

```
git branch $branch
```

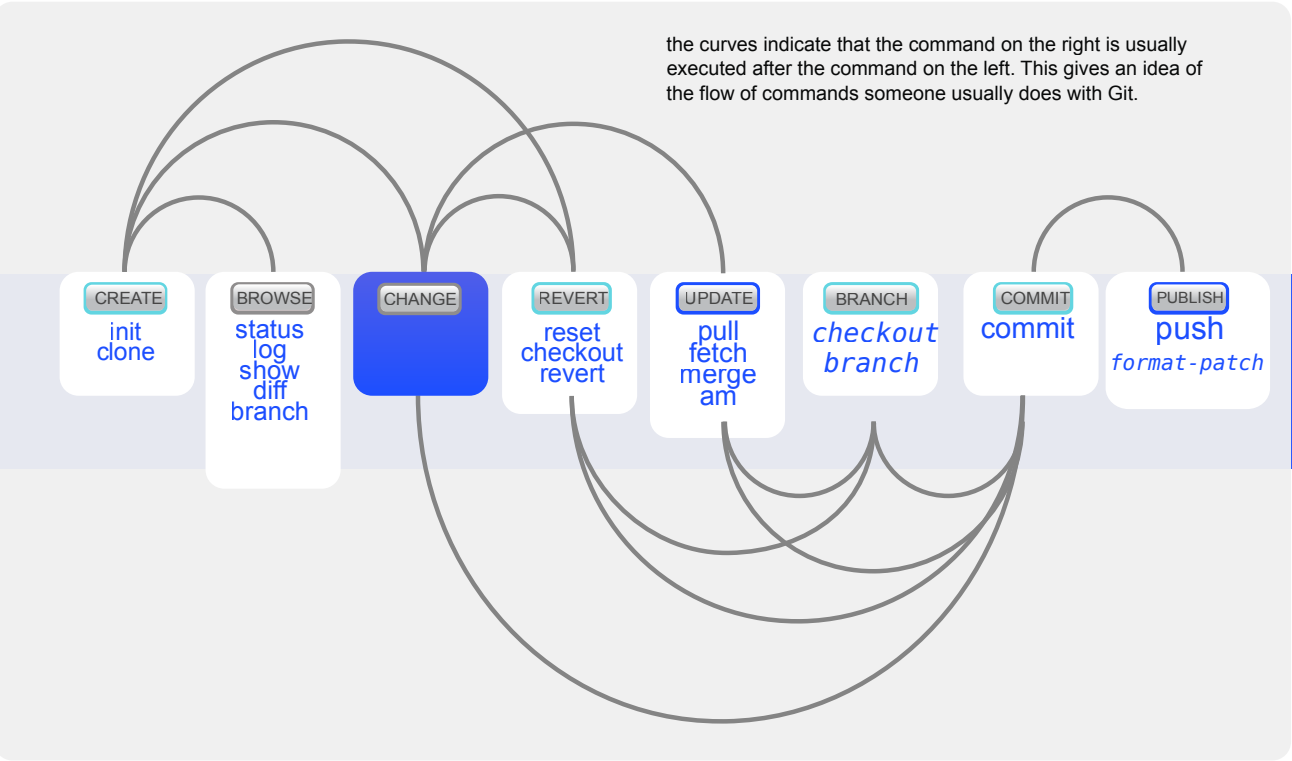
Create branch \$new\_branch based on branch \$other and switch to it

```
git checkout -b $new_branch $other
```

Delete branch \$branch

```
git branch -d $branch
```

## Commands Sequence



the curves indicate that the command on the right is usually executed after the command on the left. This gives an idea of the flow of commands someone usually does with Git.

## Update

Fetch latest changes from origin

```
git fetch
```

(but this does not merge them).

Pull latest changes from origin

```
git pull
```

(does a fetch followed by a merge)

Apply a patch that some sent you

```
git am -3 patch.mbox
```

(in case of a conflict, resolve and use `git am --resolved`)

## Publish

Commit all your local changes

```
git commit -a
```

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push
```

Mark a version / milestone

```
git tag v1.0
```

## Useful Commands

Finding regressions

```
git bisect start
git bisect good $id
git bisect bad $id
```

(to start)  
(\$id is the last working version)  
(\$id is a broken version)

```
git bisect bad/good
git bisect visualize
git bisect reset
```

(to mark it as bad or good)  
(to launch gitk and mark it)  
(once you're done)

Check for errors and cleanup repository

```
git fsck
git gc --prune
```

Search working directory for foo()

```
git grep "foo()"
```

## Resolve Merge Conflicts

To view the merge conflicts

```
git diff
```

(complete conflict diff)

```
git diff --base $file
```

(against base file)

```
git diff --ours $file
```

(against your changes)

```
git diff --theirs $file
```

(against other changes)

To discard conflicting patch

```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge with

```
git add $conflicting_file
git rebase --continue
```

(do for all resolved files)

## Cheat Sheet Notation

\$id : notation used in this sheet to represent either a commit id, branch or a tag name  
\$file : arbitrary file name  
\$branch : arbitrary branch name



# GitHub

# GIT CHEAT SHEET

V1.1.1

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

## INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

### GitHub for Windows

<https://windows.github.com>

### GitHub for Mac

<https://mac.github.com>

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

### Git for All Platforms

<http://git-scm.com>

## CONFIGURE TOOLING

Configure user information for all local repositories

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

Sets the name you want attached to your commit transactions

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

Sets the email you want attached to your commit transactions

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

Enables helpful colorization of command line output

## CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```

Creates a new local repository with the specified name

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

Downloads a project and its entire version history

## MAKE CHANGES

Review edits and craft a commit transaction

```
$ git status
```

Lists all new or modified files to be committed

```
$ git diff
```

Shows file differences not yet staged

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

Snapshots the file in preparation for versioning

```
$ git diff --staged
```

Shows file differences between staging and the last file version

```
$ git reset [file]
```

Unstages the file, but preserve its contents

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

Records file snapshots permanently in version history

## GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ git branch
```

Lists all local branches in the current repository

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

Creates a new branch

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

Switches to the specified branch and updates the working directory

```
$ git merge [branch]
```

Combines the specified branch's history into the current branch

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

Deletes the specified branch



# GIT CHEAT SHEET

## REFACTOR FILENAMES

Relocate and remove versioned files

```
$ git rm [file]
```

Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```

Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```

Changes the file name and prepares it for commit

## SUPPRESS TRACKING

Exclude temporary files and paths

```
*.log  
build/  
temp-*
```

A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```

Lists all ignored files in this project

## SAVE FRAGMENTS

Shelve and restore incomplete changes

```
$ git stash
```

Temporarily stores all modified tracked files

```
$ git stash pop
```

Restores the most recently stashed files

```
$ git stash list
```

Lists all stashed changesets

```
$ git stash drop
```

Discards the most recently stashed changeset

## REVIEW HISTORY

Browse and inspect the evolution of project files

```
$ git log
```

Lists version history for the current branch

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

Lists version history for a file, including renames

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

Shows content differences between two branches

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

Outputs metadata and content changes of the specified commit

## REDO COMMITS

Erase mistakes and craft replacement history

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

Undoes all commits after [commit], preserving changes locally

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

Discards all history and changes back to the specified commit

## SYNCHRONIZE CHANGES

Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```

Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```

Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```

Uploads all local branch commits to GitHub

```
$ git pull
```

Downloads bookmark history and incorporates changes

## GitHub Training

Learn more about using GitHub and Git. Email the Training Team or visit our web site for learning event schedules and private class availability.

✉ [training@github.com](mailto:training@github.com)  
🌐 [training.github.com](https://training.github.com)



# Git Cheat Sheet

For more awesome cheat sheets  
visit [rebellabs.org](https://rebellabs.org)!



## Create a Repository

From scratch -- Create a new local repository

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

Download from an existing repository

```
$ git clone my_url
```

## Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

## Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my\_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new\_branch

```
$ git branch new_branch
```

Delete the branch called my\_branch

```
$ git branch -d my_branch
```

Merge branch\_a into branch\_b

```
$ git checkout branch_b
```

```
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

## Make a change

Stages the file, ready for commit

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

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$ git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

## Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

```
$ git push
```

## Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit <https://training.github.com/> for official GitHub training.

