```
$ curl cheat.sh/
 cheat.sheets:git
# Set your identity.
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com

# Set your editor.
git config --global core.editor emacs

# Enable color support for commands like `git diff`. Disable with `never` or
# partially disable -- unless otherwise applied -- with `false`.
git config --global color.ui true

# Stage all changes for commit.
git add [--all|-A]

# Stash changes locally. This will keep the changes in a separate changelist, -
# called 'stash', and the working directory is cleaned. You can apply changes
# from the stash at any time.
git stash

# Stash changes with a message.
git stash save "message"

# List all the stashed changes.
git stash list
```

```
# Apply the most recent change and remove the stash from the stash list.
git stash pop

# Apply stash from the stash list, but does not remove the stash from the list.
git stash apply stash@{6}

# Commit staged changes.
git commit -m "Your commit message"

# Edit previous commit message.
git commit --amend

# Commit in the past. Newer versions of Git allow `--date="2 days ago"` usage.
git commit --date="`date --date='2 day ago'`"
git commit --date="Jun 13 18:30:25 IST 2015"

# Change the date of an existing commit.
git filter-branch --env-filter \
   'if [ $GIT_COMMIT = 119f9ecf58069b265ab22f1f97d2b648faf932e0 ]
    then
       export GIT_AUTHOR_DATE="Fri Jan 2 21:38:53 2009 -0800"
       export GIT_COMMITTER_DATE="Sat May 19 01:01:01 2007 -0700"
    fi'

# Remove staged and working directory changes.
```

```
git reset --hard

# Go 2 commits back.
git reset --hard HEAD~2

# Remove untracked files.
git clean -f -d

# Remove untracked and ignored files.
git clean -f -d -x

# Push to the tracked master branch.
git push origin master

# Push to a specified repository.
git push git@github.com:[USER_NAME]/[REPO_NAME].git

# Delete the branch "branch_name".
git branch -D [BRANCH]

# Make an existing branch track a remote branch.
git branch -u upstream/foo

# List all local and remote branches.
git branch -a
```

See who committed which line in a file.

git blame [FILE]

Sync a fork with the master repo.

git remote add upstream git@github.com:name/repo.git # <-- Set a new repo.

git remote -v # <-- Confirm new remote repo.

git fetch upstream # <-- Get branches.

git branch -va # <-- List local - remote branches.

git checkout master # <-- Checkout local master branch.

git checkout -b new_branch # <-- Create and checkout a new branch.

git merge upstream/master # <-- Merge remote into local repo.

git show 83fb499 # <-- Show what a commit did.

git show 83fb499:path/to/file.ext # <-- Show the file as it was in 83fb499.

git diff branch_1 branch_2 # <-- Check difference between branches.

git log # <-- Show all of the commits.

git status # <-- Show the changes from the last commit.

Display the commit history of a set of files.

git log --pretty=email --patch-with-stat --reverse --full-index -- Admin*.py > Sripts.patch

Import commits from another repo.

git --git-dir=../some_other_repo/.git format-patch -k -1 --stdout <commit SHA> | git am -3 -k

View commits which would be pushed.

git log @{u}..

```
# View changes which are new on a feature branch.
git log -p feature --not master
git diff master...feature

# Interactive rebase for the last 7 commits.
git rebase -i @~7

# Show changes to files WITHOUT considering them a part of git. This can be
# used to diff files which are not part of a git repo!
git diff --no-index path/to/file/A path/to/file/B

# Pull changes, while overwriting any local commits.
git fetch --all
git reset --hard origin/master

# Update all submodules.
git submodule update --init --recursive

# Perform a shallow clone, to only get the latest commits, which helps to save
# data (good for limited data connections) when cloning large repos.
git clone --depth 1 <remote-url>

# Unshallow a clone.
git pull --unshallow

# Create a bare branch; without any commits.
```

```
git checkout --orphan branch_name

# Checkout a new branch from a different starting point.
git checkout -b master upstream/master

# Reset local branch to upstream branch, then checkout it.
git checkout -B master upstream/master

# Remove all stale branches; ones that have been deleted on remote. So if you
# have a lot of useless branches, delete them on GitHub and then run this.
git remote prune origin

# Prune all remotes at once.
git remote prune $(git remote | tr '\n' ' ')

# Revisions can also be identified with `:/text`. So, this will show the first
# commit that has the string "cool" in its message body.
git show :/cool

# Undo parts of the last commit in a specific file.
git checkout -p HEAD^ -- /path/to/file

# Revert a commit, but keep the history of the event as a separate commit.
git revert <commit SHA>

# Apply only the changes made within a given commit. This is different to the
```

```
# `merge` command, as it would otherwise apply all commits from a branch.
git cherry-pick [HASH]

# Undo last commit. If you want to nuke commit C to never see it again:
#   (F)
# A-B-C
#    ↑
#   master
git reset --hard HEAD~1

# Undo last commit. If you want to undo the commit, but keep your changes:
#   (F)
# A-B-C
#   ↑
# master
git reset HEAD~1

# List files changed in a given commit.
git diff-tree --no-commit-id --name-only -r [HASH]

# Porcelain-ly List files changed in a given commit; user-facing approach.
git show --pretty="" --name-only bd61ad98

# See everything you have done, across branches, in a glance, then go to the
# place right before you broke everything.
git reflog
```

git reset HEAD@{hash}

Move your most recent commit from one branch, to stage it on [BRANCH].

git reset HEAD~ --soft

git stash

git checkout [BRANCH]

git stash pop

git add .

cheat:git

---

tags: [vcs]

To set your identity:

git config --global user.name <name>

git config --global user.email <email>

To set your editor:

git config --global core.editor <editor>

To enable color:

git config --global color.ui true

To stage all changes for commit:

git add --all

```
# To stash changes locally, this will keep the changes in a separate changelist
# called stash and the working directory is cleaned. You can apply changes
# from the stash anytime
git stash

# To stash changes with a message:
git stash push -m <message>

# To list all the stashed changes:
git stash list

# To apply the most recent change and remove the stash from the stash list:
git stash pop

# To apply any stash from the list of stashes. This does not remove the stash
# from the stash list
git stash apply stash@{6}

# To commit staged changes:
git commit -m <message>

# To edit previous commit message:
git commit --amend

# Git commit in the past
git commit --date="`date --date='2 day ago'`"
```

```
git commit --date="Jun 13 18:30:25 IST 2015"
# more recent versions of Git also support --date="2 days ago" directly

# To change the date of an existing commit:
git filter-branch --env-filter \
   'if [ $GIT_COMMIT = 119f9ecf58069b265ab22f1f97d2b648faf932e0 ]
    then
       export GIT_AUTHOR_DATE="Fri Jan 2 21:38:53 2009 -0800"
       export GIT_COMMITTER_DATE="Sat May 19 01:01:01 2007 -0700"
    fi'

# To remove staged and working directory changes:
git reset --hard

# To go 2 commits back:
git reset --hard HEAD~2

# Checkout the fb branch, and rebase from <remote>
git reset --hard <remote>/<branch>

# To revert first/initial commit on a branch:
# Running git reset --hard HEAD~1 will give error:
# fatal: ambiguous argument 'HEAD~1': unknown revision or path not in the working tree.
git update-ref -d HEAD

# To remove untracked files:
```

```
git clean -f -d

# To remove untracked and ignored files:
git clean -f -d -x

# To push to the tracked master branch:
git push origin master

# To push to a specified repository:
git push git@github.com:<username>/<repo>.git

# Tags: Tag a commit
git tag -a <tag> <commit> -m "<commit message>"

# Tags: To push a tag to remote:
git push origin <tagname>

# Tags: To delete a tag <tagname> on remote
git push --delete origin <tagname>

# Tags: To delete a tag locally
git tag -d <tagname>

# To force a push:
git push -f
```

Branches: To delete the branch <branch>:

git branch -D <branch>

Branches: To delete a local <branch>:

git branch -d <branch>

Branches: To delete a remote branch <branch>:

git push --delete origin <branch>

Branches: To delete all branches on remote that are already merged:

git branch --merged | egrep -v "(^*|main|dev)" | xargs git branch -d

Branches: To make an exisiting branch track a remote branch:

git branch -u upstream/foo

To see who commited which line in a file:

git blame <file>

To sync a fork with the master repo:

git remote add upstream git@github.com:<username>/<repo>.git # Set a new repo

git remote -v # Confirm new remote repo

git fetch upstream # Get branches

git branch -va # List local - remote branches

git checkout master # Checkout local master branch

git checkout -b new_branch # Create and checkout a new branch

git merge upstream/master # Merge remote into local repo

```
git show 83fb499                        # Show what a commit did.
git show 83fb499:path/fo/file.ext       # Shows the file as it appeared at 83fb499.
git diff branch_1 branch_2              # Check difference between branches
git log                                 # Show all the commits
git status                              # Show the changes from last commit

# To view the commit history of a set of files:
git log --pretty=email --patch-with-stat --reverse --full-index -- Admin\*.py > Sripts.patch

# To import commits from another repo:
git --git-dir=../some_other_repo/.git format-patch -k -1 --stdout <commit SHA> | git am -3 -k

# To view commits that will be pushed:
git log @{u}..

# To view changes that are new on a feature branch:
git log -p feature --not master
git diff master...feature

# To perform an interactive rebase for the prior 7 commits:
git rebase -i @~7

# To diff files WITHOUT considering them a part of git:
# This can be used to diff files that are not in a git repo!
git diff --no-index path/to/file/A path/to/file/B
```

```
# To pull changes while overwriting any local commits:
git fetch --all
git reset --hard origin/master

# To pull down a remote branch, but rebase any locally differing commits onto
# the top of the incoming commits:
git pull <remote> <branch> --rebase

# To update all submodules:
git submodule update --init --recursive

# To perform a shallow clone to only get latest commits:
# (helps save data when cloning large repos)
git clone --depth 1 <remote-url>

# To unshallow a clone:
git pull --unshallow

# To create a bare branch (one that has no commits on it):
git checkout --orphan branch_name

# To checkout a new branch from a different starting point:
git checkout -b master upstream/master

# To remove all stale branches (ones that have been deleted on remote): So if
# you have a lot of useless branches, delete them on Github and then run this:
```

```
git remote prune origin

# To prune all remotes at once:
git remote prune $(git remote | tr '\n' ' ')

# Revisions can also be identified with :/text
# So, this will show the first commit that has "cool" in their message body
git show :/cool

# To undo parts of last commit in a specific file:
git checkout -p HEAD^ -- /path/to/file

# To revert a commit and keep the history of the reverted change as a separate revert commit:
git revert <commit SHA>

# To pick a commit from a branch to current branch. This is different than
# merge as this just applies a single commit from a branch to current branch:
git cherry-pick <commit SHA1>

# Change author of a commit:
git commit --amend --author="Author Name <email@address.com>"

# The GPG key used for signing your commits
git config --global user.signingkey 0A46826A
```

Sign new tags:

git tag -s v1.5 -m 'my signed 1.5 tag'

Sign a commit:

git commit -a -S -m 'Signed commit'

check any signatures it finds and list them in its output:

git log --pretty="format:%h %G? %aN %s"

Defined the key to use for signing commits:

git config user.signingkey [KEYID]

Set signing of commits globally:

git config --global commit.gpgsign true

To list unstracked files:

git ls-files --others --exclude-standard

tldr:git

git

Distributed version control system.

Some subcommands such as `commit`, `add`, `branch`, `checkout`, `push`, etc. have their own usage documentation, accessible via `tldr git subcommand`.

More information: <https://git-scm.com/>.

Check the Git version:

```
git --version

# Show general help:
git --help

# Show help on a Git subcommand (like `clone`, `add`, `push`, `log`, etc.):
git help subcommand

# Execute a Git subcommand:
git subcommand

# Execute a Git subcommand on a custom repository root path:
git -C path/to/repo subcommand

# Execute a Git subcommand with a given configuration set:
git -c 'config.key=value' subcommand

$
$ curl cheat.sh/
 cheat.sheets:git
# Set your identity.
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com

# Set your editor.
git config --global core.editor emacs
```

```
# Enable color support for commands like `git diff`. Disable with `never` or
# partially disable -- unless otherwise applied -- with `false`.
git config --global color.ui true

# Stage all changes for commit.
git add [--all|-A]

# Stash changes locally. This will keep the changes in a separate changelist, -
# called 'stash', and the working directory is cleaned. You can apply changes
# from the stash at any time.
git stash

# Stash changes with a message.
git stash save "message"

# List all the stashed changes.
git stash list

# Apply the most recent change and remove the stash from the stash list.
git stash pop

# Apply stash from the stash list, but does not remove the stash from the list.
git stash apply stash@{6}

# Commit staged changes.
```

```
git commit -m "Your commit message"

# Edit previous commit message.
git commit --amend

# Commit in the past. Newer versions of Git allow `--date="2 days ago"` usage.
git commit --date="`date --date='2 day ago'`"
git commit --date="Jun 13 18:30:25 IST 2015"

# Change the date of an existing commit.
git filter-branch --env-filter \
   'if [ $GIT_COMMIT = 119f9ecf58069b265ab22f1f97d2b648faf932e0 ]
    then
       export GIT_AUTHOR_DATE="Fri Jan 2 21:38:53 2009 -0800"
       export GIT_COMMITTER_DATE="Sat May 19 01:01:01 2007 -0700"
    fi'

# Remove staged and working directory changes.
git reset --hard

# Go 2 commits back.
git reset --hard HEAD~2

# Remove untracked files.
git clean -f -d
```

```
# Remove untracked and ignored files.
git clean -f -d -x

# Push to the tracked master branch.
git push origin master

# Push to a specified repository.
git push git@github.com:[USER_NAME]/[REPO_NAME].git

# Delete the branch "branch_name".
git branch -D [BRANCH]

# Make an existing branch track a remote branch.
git branch -u upstream/foo

# List all local and remote branches.
git branch -a

# See who committed which line in a file.
git blame [FILE]

# Sync a fork with the master repo.
git remote add upstream git@github.com:name/repo.git # <-- Set a new repo.
git remote -v # <-- Confirm new remote repo.
git fetch upstream # <-- Get branches.
git branch -va # <-- List local - remote branches.
```

git checkout master # <-- Checkout local master branch.

git checkout -b new_branch # <-- Create and checkout a new branch.

git merge upstream/master # <-- Merge remote into local repo.

git show 83fb499 # <-- Show what a commit did.

git show 83fb499:path/to/file.ext # <-- Show the file as it was in 83fb499.

git diff branch_1 branch_2 # <-- Check difference between branches.

git log # <-- Show all of the commits.

git status # <-- Show the changes from the last commit.

Display the commit history of a set of files.

git log --pretty=email --patch-with-stat --reverse --full-index -- Admin*.py > Sripts.patch

Import commits from another repo.

git --git-dir=../some_other_repo/.git format-patch -k -1 --stdout <commit SHA> | git am -3 -k

View commits which would be pushed.

git log @{u}..

View changes which are new on a feature branch.

git log -p feature --not master

git diff master...feature

Interactive rebase for the last 7 commits.

git rebase -i @~7

Show changes to files WITHOUT considering them a part of git. This can be

```
# used to diff files which are not part of a git repo!
git diff --no-index path/to/file/A path/to/file/B

# Pull changes, while overwriting any local commits.
git fetch --all
git reset --hard origin/master

# Update all submodules.
git submodule update --init --recursive

# Perform a shallow clone, to only get the latest commits, which helps to save
# data (good for limited data connections) when cloning large repos.
git clone --depth 1 <remote-url>

# Unshallow a clone.
git pull --unshallow

# Create a bare branch; without any commits.
git checkout --orphan branch_name

# Checkout a new branch from a different starting point.
git checkout -b master upstream/master

# Reset local branch to upstream branch, then checkout it.
git checkout -B master upstream/master
```

```
# Remove all stale branches; ones that have been deleted on remote. So if you
# have a lot of useless branches, delete them on GitHub and then run this.
git remote prune origin

# Prune all remotes at once.
git remote prune $(git remote | tr '\n' ' ')

# Revisions can also be identified with `:/text`. So, this will show the first
# commit that has the string "cool" in its message body.
git show :/cool

# Undo parts of the last commit in a specific file.
git checkout -p HEAD^ -- /path/to/file

# Revert a commit, but keep the history of the event as a separate commit.
git revert <commit SHA>

# Apply only the changes made within a given commit. This is different to the
# `merge` command, as it would otherwise apply all commits from a branch.
git cherry-pick [HASH]

# Undo last commit. If you want to nuke commit C to never see it again:
#   (F)
# A-B-C
#    ↑
#  master
```

```
git reset --hard HEAD~1

# Undo last commit. If you want to undo the commit, but keep your changes:
#   (F)
# A-B-C
#  ↑
# master
git reset HEAD~1

# List files changed in a given commit.
git diff-tree --no-commit-id --name-only -r [HASH]

# Porcelain-ly List files changed in a given commit; user-facing approach.
git show --pretty="" --name-only bd61ad98

# See everything you have done, across branches, in a glance, then go to the
# place right before you broke everything.
git reflog
git reset HEAD@{hash}

# Move your most recent commit from one branch, to stage it on [BRANCH].
git reset HEAD~ --soft
git stash
git checkout [BRANCH]
git stash pop
git add .
```

cheat:git

---

tags: [vcs]

To set your identity:

git config --global user.name <name>

git config --global user.email <email>

To set your editor:

git config --global core.editor <editor>

To enable color:

git config --global color.ui true

To stage all changes for commit:

git add --all

To stash changes locally, this will keep the changes in a separate changelist

called stash and the working directory is cleaned. You can apply changes

from the stash anytime

git stash

To stash changes with a message:

git stash push -m <message>

```
# To list all the stashed changes:
git stash list

# To apply the most recent change and remove the stash from the stash list:
git stash pop

# To apply any stash from the list of stashes. This does not remove the stash
# from the stash list
git stash apply stash@{6}

# To commit staged changes:
git commit -m <message>

# To edit previous commit message:
git commit --amend

# Git commit in the past
git commit --date="`date --date='2 day ago'`"
git commit --date="Jun 13 18:30:25 IST 2015"
# more recent versions of Git also support --date="2 days ago" directly

# To change the date of an existing commit:
git filter-branch --env-filter \
   'if [ $GIT_COMMIT = 119f9ecf58069b265ab22f1f97d2b648faf932e0 ]
    then
       export GIT_AUTHOR_DATE="Fri Jan 2 21:38:53 2009 -0800"
```

```
    export GIT_COMMITTER_DATE="Sat May 19 01:01:01 2007 -0700"
  fi'

# To remove staged and working directory changes:
git reset --hard

# To go 2 commits back:
git reset --hard HEAD~2

# Checkout the fb branch, and rebase from <remote>
git reset --hard <remote>/<branch>

# To revert first/initial commit on a branch:
# Running git reset --hard HEAD~1 will give error:
# fatal: ambiguous argument 'HEAD~1': unknown revision or path not in the working tree.
git update-ref -d HEAD

# To remove untracked files:
git clean -f -d

# To remove untracked and ignored files:
git clean -f -d -x

# To push to the tracked master branch:
git push origin master
```

To push to a specified repository:

git push git@github.com:<username>/<repo>.git

Tags: Tag a commit

git tag -a <tag> <commit> -m "<commit message>"

Tags: To push a tag to remote:

git push origin <tagname>

Tags: To delete a tag <tagname> on remote

git push --delete origin <tagname>

Tags: To delete a tag locally

git tag -d <tagname>

To force a push:

git push -f

Branches: To delete the branch <branch>:

git branch -D <branch>

Branches: To delete a local <branch>:

git branch -d <branch>

Branches: To delete a remote branch <branch>:

git push --delete origin <branch>

Branches: To delete all branches on remote that are already merged:

git branch --merged | egrep -v "(^*|main|dev)" | xargs git branch -d

Branches: To make an exisiting branch track a remote branch:

git branch -u upstream/foo

To see who commited which line in a file:

git blame <file>

To sync a fork with the master repo:

git remote add upstream git@github.com:<username>/<repo>.git # Set a new repo

git remote -v # Confirm new remote repo

git fetch upstream # Get branches

git branch -va # List local - remote branches

git checkout master # Checkout local master branch

git checkout -b new_branch # Create and checkout a new branch

git merge upstream/master # Merge remote into local repo

git show 83fb499 # Show what a commit did.

git show 83fb499:path/fo/file.ext # Shows the file as it appeared at 83fb499.

git diff branch_1 branch_2 # Check difference between branches

git log # Show all the commits

git status # Show the changes from last commit

To view the commit history of a set of files:

git log --pretty=email --patch-with-stat --reverse --full-index -- Admin*.py > Sripts.patch

To import commits from another repo:

git --git-dir=../some_other_repo/.git format-patch -k -1 --stdout <commit SHA> | git am -3 -k

To view commits that will be pushed:

git log @{u}..

To view changes that are new on a feature branch:

git log -p feature --not master

git diff master...feature

To perform an interactive rebase for the prior 7 commits:

git rebase -i @~7

To diff files WITHOUT considering them a part of git:

This can be used to diff files that are not in a git repo!

git diff --no-index path/to/file/A path/to/file/B

To pull changes while overwriting any local commits:

git fetch --all

git reset --hard origin/master

To pull down a remote branch, but rebase any locally differing commits onto

the top of the incoming commits:

git pull <remote> <branch> --rebase

```
# To update all submodules:
git submodule update --init --recursive

# To perform a shallow clone to only get latest commits:
# (helps save data when cloning large repos)
git clone --depth 1 <remote-url>

# To unshallow a clone:
git pull --unshallow

# To create a bare branch (one that has no commits on it):
git checkout --orphan branch_name

# To checkout a new branch from a different starting point:
git checkout -b master upstream/master

# To remove all stale branches (ones that have been deleted on remote): So if
# you have a lot of useless branches, delete them on Github and then run this:
git remote prune origin

# To prune all remotes at once:
git remote prune $(git remote | tr '\n' ' ')

# Revisions can also be identified with :/text
# So, this will show the first commit that has "cool" in their message body
git show :/cool
```

To undo parts of last commit in a specific file:

git checkout -p HEAD^ -- /path/to/file

To revert a commit and keep the history of the reverted change as a separate revert commit:

git revert <commit SHA>

To pick a commit from a branch to current branch. This is different than

merge as this just applies a single commit from a branch to current branch:

git cherry-pick <commit SHA1>

Change author of a commit:

git commit --amend --author="Author Name <email@address.com>"

The GPG key used for signing your commits

git config --global user.signingkey 0A46826A

Sign new tags:

git tag -s v1.5 -m 'my signed 1.5 tag'

Sign a commit:

git commit -a -S -m 'Signed commit'

check any signatures it finds and list them in its output:

git log --pretty="format:%h %G? %aN %s"

Defined the key to use for signing commits:

git config user.signingkey [KEYID]

Set signing of commits globally:

git config --global commit.gpgsign true

To list unstracked files:

git ls-files --others --exclude-standard

tldr:git

git

Distributed version control system.

Some subcommands such as `commit`, `add`, `branch`, `checkout`, `push`, etc. have their own usage documentation, accessible via `tldr git subcommand`.

More information: <https://git-scm.com/>.

Check the Git version:

git --version

Show general help:

git --help

Show help on a Git subcommand (like `clone`, `add`, `push`, `log`, etc.):

git help subcommand

Execute a Git subcommand:

git subcommand

Execute a Git subcommand on a custom repository root path:

git -C path/to/repo subcommand

Execute a Git subcommand with a given configuration set:

git -c 'config.key=value' subcommand

$

v