CronJob

Create a cron job with the specified name.

Examples:
```
  # Create a cron job
  kubectl create cronjob my-job --image=busybox --schedule="*/1 * * * *"

  # Create a cron job with a command
  kubectl create cronjob my-job --image=busybox --schedule="*/1 * * * *" -- date
```

Options:

--dry-run='none':
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

--field-manager='kubectl-create':
Name of the manager used to track field ownership.

--image='':
Image name to run.

-o, --output='':
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

--restart='':
job's restart policy. supported values: OnFailure, Never

--save-config=false:
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

--schedule='':
A schedule in the Cron format the job should be run with.

Usage:
```
  kubectl create cronjob NAME --image=image --schedule='0/5 * * * ?' -- [COMMAND] [args...] [flags] [options]
```

Job

Create a job with the specified name.

Examples:
```
  # Create a job
  kubectl create job my-job --image=busybox

  # Create a job with a command
  kubectl create job my-job --image=busybox -- date

  # Create a job from a cron job named "a-cronjob"
  kubectl create job test-job --from=cronjob/a-cronjob
```

Options:

--dry-run='none':
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

--field-manager='kubectl-create':
Name of the manager used to track field ownership.

--from='':
The name of the resource to create a Job from (only cronjob is supported).

--image='':
Image name to run.

-o, --output='':
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

--save-config=false:
If true, the configuration of current object will be saved in its annotation. Otherwise, the annotation will be unchanged. This flag is useful when you want to perform kubectl apply on this object in the future.

--show-managed-fields=false:
If true, keep the managedFields when printing objects in JSON or YAML format.

Usage:
```
  kubectl create job NAME --image=image [--from=cronjob/name] -- [COMMAND] [args...] [options]
```

Ingress

Create an ingress with the specified name.

Examples:

```
  # Create a single ingress called 'simple' that directs requests to foo.com/bar to svc
  kubectl create ingress simple --rule="foo.com/bar=svc1:8080,tls=my-cert"

  # Create a catch all ingress of "/path" pointing to service svc:port and Ingress Class as "otheringress"
  kubectl create ingress catch-all --class=otheringress --rule="/path=svc:port"

  # Create an ingress with two annotations: ingress.annotation1 and ingress.annotations2
  kubectl create ingress annotated --class=default --rule="foo.com/bar=svc:port" \
  --annotation ingress.annotation1=foo \
  --annotation ingress.annotation2=bla

  # Create an ingress with the same host and multiple paths
  kubectl create ingress multipath --class=default \
  --rule="foo.com/=svc:port" \
  --rule="foo.com/admin/=svcadmin:portadmin"

  # Create an ingress with multiple hosts and the pathType as Prefix
  kubectl create ingress ingress1 --class=default \
  --rule="foo.com/path*=svc:8080" \
  --rule="bar.com/admin*=svc2:http"

  # Create an ingress with TLS enabled using the default ingress certificate and different path types
  kubectl create ingress ingtls --class=default \
  --rule="foo.com/=svc:https,tls" \
  --rule="foo.com/path/subpath*=othersvc:8080"

  # Create an ingress with TLS enabled using a specific secret and pathType as Prefix
  kubectl create ingress ingsecret --class=default \
  --rule="foo.com/*=svc:8080,tls=secret1"

  # Create an ingress with a default backend
  kubectl create ingress ingdefault --class=default \
  --default-backend=defaultsvc:http \
  --rule="foo.com/*=svc:8080,tls=secret1"
```

Options:

```
        --annotation=[]:
    Annotation to insert in the ingress object, in the format annotation=value

        --class='':
    Ingress Class to be used

        --default-backend='':
    Default service for backend, in format of svcname:port

        --rule=[]:
    Rule in format host/path=service:port[,tls=secretname]. Paths containing the leading character '*' are
considered pathType=Prefix. tls argument is optional.
```

Usage:

```
  kubectl create ingress NAME --rule=host/path=service:port[,tls[=secret]]  [options]
```

Label

Update the labels on a resource.

Examples:
```
  # Update pod 'foo' with the label 'unhealthy' and the value 'true'
  kubectl label pods foo unhealthy=true

  # Update pod 'foo' with the label 'status' and the value 'unhealthy', overwriting any existing value
  kubectl label --overwrite pods foo status=unhealthy

  # Update all pods in the namespace
  kubectl label pods --all status=unhealthy

  # Update a pod identified by the type and name in "pod.json"
  kubectl label -f pod.json status=unhealthy

  # Update pod 'foo' only if the resource is unchanged from version 1
  kubectl label pods foo status=unhealthy --resource-version=1

  # Update pod 'foo' by removing a label named 'bar' if it exists
  # Does not require the --overwrite flag
  kubectl label pods foo bar-
```

Options:

--all=false:
Select all resources, in the namespace of the specified resource types

-A, --all-namespaces=false:
If true, check the specified action in all namespaces.

--field-selector='':
Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.

-f, --filename=[]:
Filename, directory, or URL to files identifying the resource to update the labels

--list=false:
If true, display the labels for a given resource.

--local=false:
If true, label will NOT contact api-server but run locally.

--overwrite=false:
If true, allow labels to be overwritten, otherwise reject label updates that overwrite existing labels.

-l, --selector='':
Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2). Matching objects must satisfy all of the specified label constraints.

Usage:
```
  kubectl label [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N
[--resource-version=version] [options]
```

Annotate

Update the annotations on one or more resources.

Examples:
```
  # Update pod 'foo' with the annotation 'description' and the value 'my frontend'
  # If the same annotation is set multiple times, only the last value will be applied
  kubectl annotate pods foo description='my frontend'

  # Update a pod identified by type and name in "pod.json"
  kubectl annotate -f pod.json description='my frontend'

  # Update pod 'foo' with the annotation 'description' and the value 'my frontend running nginx', overwriting any existing value
  kubectl annotate --overwrite pods foo description='my frontend running nginx'

  # Update all pods in the namespace
  kubectl annotate pods --all description='my frontend running nginx'

  # Update pod 'foo' only if the resource is unchanged from version 1
  kubectl annotate pods foo description='my frontend running nginx' --resource-version=1

  # Update pod 'foo' by removing an annotation named 'description' if it exists
  kubectl annotate pods foo description-
```

Options:

--all=false:
Select all resources, in the namespace of the specified resource types.

--field-manager='kubectl-annotate':
Name of the manager used to track field ownership.

--field-selector='':
Selector (field query) to filter on, supports '=', '==', and '!='.(e.g. --field-selector key1=value1,key2=value2). The server only supports a limited number of field queries per type.

-f, --filename=[]:
Filename, directory, or URL to files identifying the resource to update the annotation

--list=false:
If true, display the annotations for a given resource.

--local=false:
If true, annotation will NOT contact api-server but run locally.

--overwrite=false:
If true, allow annotations to be overwritten, otherwise reject annotation updates that overwrite existing annotations.

-l, --selector='':
Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2). Matching objects must satisfy all of the specified label constraints.

Usage:
```
  kubectl annotate [--overwrite] (-f FILENAME | TYPE NAME) KEY_1=VAL_1 ... KEY_N=VAL_N [--resource-version=version] [options]
```

ConfigMap

Create a config map based on a file, directory, or specified literal value.

Examples:

```
  # Create a new config map named my-config based on folder bar
  kubectl create configmap my-config --from-file=path/to/bar

  # Create a new config map named my-config with specified keys instead of file basenames on disk
  kubectl create configmap my-config --from-file=key1=/path/to/bar/file1.txt --from-file=key2=/path/to/bar/file2.txt

  # Create a new config map named my-config with key1=config1 and key2=config2
  kubectl create configmap my-config --from-literal=key1=config1 --from-literal=key2=config2

  # Create a new config map named my-config from the key=value pairs in the file
  kubectl create configmap my-config --from-file=path/to/bar

  # Create a new config map named my-config from an env file
  kubectl create configmap my-config --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

Options:

--allow-missing-template-keys=true:
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

--append-hash=false:
Append a hash of the configmap to its name.

--from-env-file=[]:
Specify the path to a file to read lines of key=val pairs to create a configmap.

--from-file=[]:
Key file can be specified using its file path, in which case file basename will be used as configmap key, or optionally with a key and file path, in which case the given key will be used. Specifying a directory will iterate each named file in the directory whose basename is a valid configmap key.

--from-literal=[]:
Specify a key and literal value to insert in configmap (i.e. mykey=somevalue)

Usage:

```
  kubectl create configmap NAME [--from-file=[key=]source] [--from-literal=key1=value1]
[--dry-run=server|client|none] [options]
```

Deployment

Create a deployment with the specified name.

Examples:
```
  # Create a deployment named my-dep that runs the busybox image
  kubectl create deployment my-dep --image=busybox

  # Create a deployment with a command
  kubectl create deployment my-dep --image=busybox -- date

  # Create a deployment named my-dep that runs the nginx image with 3 replicas
  kubectl create deployment my-dep --image=nginx --replicas=3

  # Create a deployment named my-dep that runs the busybox image and expose port 5701
  kubectl create deployment my-dep --image=busybox --port=5701
```

Options:

--image=[]:
Image names to run.

-o, --output='':
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

--port=-1:
The port that this container exposes.

-r, --replicas=1:
Number of replicas to create. Default is 1.

Usage:
```
  kubectl create deployment NAME --image=image -- [COMMAND] [args...] [options]
```

Secret Generic

Create a secret based on a file, directory, or specified literal value.

Examples:
```
  # Create a new secret named my-secret with keys for each file in folder bar
  kubectl create secret generic my-secret --from-file=path/to/bar

  # Create a new secret named my-secret with specified keys instead of names on disk
  kubectl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa
--from-file=ssh-publickey=path/to/id_rsa.pub

  # Create a new secret named my-secret with key1=supersecret and key2=topsecret
  kubectl create secret generic my-secret --from-literal=key1=supersecret --from-literal=key2=topsecret

  # Create a new secret named my-secret using a combination of a file and a literal
  kubectl create secret generic my-secret --from-file=ssh-privatekey=path/to/id_rsa
--from-literal=passphrase=topsecret

  # Create a new secret named my-secret from env files
  kubectl create secret generic my-secret --from-env-file=path/to/foo.env --from-env-file=path/to/bar.env
```

Options:

--from-env-file=[]:
Specify the path to a file to read lines of key=val pairs to create a secret.

--from-file=[]:
Key files can be specified using their file path, in which case a default name will be given to them, or optionally with a name and file path, in which case the given name will be used. Specifying a directory will iterate each named file in the directory that is a valid secret key.

--from-literal=[]:
Specify a key and literal value to insert in secret (i.e. mykey=somevalue)

--type='':
The type of secret to create

Usage:
```
  kubectl create secret generic NAME [--type=string] [--from-file=[key=]source] [--from-literal=key1=value1]
[--dry-run=server|client|none] [options]
```

Secret Docker Registry

Create a new secret for use with Docker registries.

When using the Docker command line to push images, you can authenticate to a given registry by running:
'$ docker login DOCKER_REGISTRY_SERVER --username=DOCKER_USER
--password=DOCKER_PASSWORD --email=DOCKER_EMAIL'.

That produces a ~/.dockercfg file that is used by subsequent 'docker push' and 'docker pull' commands to authenticate to the registry. The email address is optional.

Examples:

```
# If you don't already have a .dockercfg file, you can create a dockercfg secret directly by using:
kubectl create secret docker-registry my-secret --docker-server=DOCKER_REGISTRY_SERVER
--docker-username=DOCKER_USER --docker-password=DOCKER_PASSWORD
--docker-email=DOCKER_EMAIL

# Create a new secret named my-secret from ~/.docker/config.json
kubectl create secret docker-registry my-secret --from-file=.dockerconfigjson=path/to/.docker/config.json
```

Options:

```
        --docker-email='':
    Email for Docker registry

        --docker-password='':
    Password for Docker registry authentication

        --docker-server='https://index.docker.io/v1/':
    Server location for Docker registry

        --docker-username='':
    Username for Docker registry authentication

        --from-file=[]:
    Key files can be specified using their file path, in which case a default name will be given to them, or optionally
with a name and file path, in which case the given name will be used.  Specifying a directory will iterate each
named file in the directory that is a valid secret key.
```

Usage:

```
kubectl create secret docker-registry NAME --docker-username=user --docker-password=password
--docker-email=email [--docker-server=string] [--from-file=[key=]source] [--dry-run=server|client|none] [options]
```

Autoscale

Creates an autoscaler that automatically chooses and sets the number of pods that run in a Kubernetes cluster.

Examples:
```
  # Auto scale a deployment "foo", with the number of pods between 2 and 10, no target CPU utilization specified so a default autoscaling policy will be used
  kubectl autoscale deployment foo --min=2 --max=10

  # Auto scale a replication controller "foo", with the number of pods between 1 and 5, target CPU utilization at 80%
  kubectl autoscale rc foo --max=5 --cpu-percent=80
```

Options:

--cpu-percent=-1:
The target average CPU utilization (represented as a percent of requested CPU) over all the pods. If it's not specified or negative, a default autoscaling policy will be used.

-f, --filename=[]:
Filename, directory, or URL to files identifying the resource to autoscale.

--max=-1:
The upper limit for the number of pods that can be set by the autoscaler. Required.

--min=-1:
The lower limit for the number of pods that can be set by the autoscaler. If it's not specified or negative, the server will apply a default value.

--name='':
The name for the newly created object. If not specified, the name of the input resource will be used.

Usage:
```
  kubectl autoscale (-f FILENAME | TYPE NAME | TYPE/NAME) [--min=MINPODS] --max=MAXPODS [--cpu-percent=CPU] [options]
```

Role

Create a role with single rule.

Examples:
```
  # Create a role named "pod-reader" that allows user to perform "get", "watch" and "list" on pods
  kubectl create role pod-reader --verb=get --verb=list --verb=watch --resource=pods

  # Create a role named "pod-reader" with ResourceName specified
  kubectl create role pod-reader --verb=get --resource=pods --resource-name=readablepod
--resource-name=anotherpod

  # Create a role named "foo" with API Group specified
  kubectl create role foo --verb=get,list,watch --resource=rs.apps

  # Create a role named "foo" with SubResource specified
  kubectl create role foo --verb=get,list,watch --resource=pods,pods/status
```

Options:
```
        --resource=[]:
    Resource that the rule applies to

        --resource-name=[]:
    Resource in the white list that the rule applies to, repeat this flag for multiple items

        --verb=[]:
    Verb that applies to the resources contained in the rule
```

Usage:
```
  kubectl create role NAME --verb=verb --resource=resource.group/subresource
[--resource-name=resourcename] [--dry-run=server|client|none] [options]
```

Role Binding

Create a role binding for a particular role or cluster role.

Examples:
```
  # Create a role binding for user1, user2, and group1 using the admin cluster role
  kubectl create rolebinding admin --clusterrole=admin --user=user1 --user=user2 --group=group1
```

Options:
```
        --clusterrole='':
    ClusterRole this RoleBinding should reference

        --group=[]:
    Groups to bind to the role. The flag can be repeated to add multiple groups.

        --role='':
    Role this RoleBinding should reference

        --serviceaccount=[]:
    Service accounts to bind to the role, in the format <namespace>:<name>. The flag can be repeated to add
multiple service accounts.

        --user=[]:
    Usernames to bind to the role. The flag can be repeated to add multiple users.
```

Usage:
```
  kubectl create rolebinding NAME --clusterrole=NAME|--role=NAME [--user=username] [--group=groupname]
[--serviceaccount=namespace:serviceaccountname] [--dry-run=server|client|none] [options]
```

Service ClusterIP

Create a ClusterIP service with the specified name.

Examples:
```
  # Create a new ClusterIP service named my-cs
  kubectl create service clusterip my-cs --tcp=5678:8080

  # Create a new ClusterIP service named my-cs (in headless mode)
  kubectl create service clusterip my-cs --clusterip="None"
```

Options:
```
        --clusterip='':
    Assign your own ClusterIP or set to 'None' for a 'headless' service (no loadbalancing).
        --tcp=[]:
    Port pairs can be specified as '<port>:<targetPort>'.
```

Usage:
```
  kubectl create service clusterip NAME [--tcp=<port>:<targetPort>] [--dry-run=server|client|none] [options]
```

Run

Create and run a particular image in a pod.

Examples:

```
 # Start a hazelcast pod and set labels "app=hazelcast" and "env=prod" in the container
  kubectl run hazelcast --image=hazelcast/hazelcast --labels="app=hazelcast,env=prod"

  # Start a nginx pod, but overload the spec with a partial set of values parsed from JSON
  kubectl run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": { ... } }'

  # Start a busybox pod and keep it in the foreground, don't restart it if it exits
  kubectl run -i -t busybox --image=busybox --restart=Never

  # Start the nginx pod using the default command, but use custom arguments (arg1 .. argN) for that command
  kubectl run nginx --image=nginx -- <arg1> <arg2> ... <argN>

  # Start the nginx pod using a different command and custom arguments
  kubectl run nginx --image=nginx --command -- <cmd> <arg1> ... <argN>
```

Options:

```
        --annotations=[]:
    Annotations to apply to the pod.

        --env=[]:
    Environment variables to set in the container.

        --expose=false:
    If true, create a ClusterIP service associated with the pod.  Requires `--port`.

        --image='':
    The image for the container to run.

        --image-pull-policy='':
    The image pull policy for the container.  If left empty, this value will not be specified by the client and defaulted
by the server.

        -l, --labels='':
    Comma separated labels to apply to the pod. Will override previous values.

        --port='':
    The port that this container exposes.

        --privileged=false:
    If true, run the container in privileged mode.

        --restart='Always':
    The restart policy for this Pod.  Legal values [Always, OnFailure, Never].

        --rm=false:
    If true, delete the pod after it exits.  Only valid when attaching to the container, e.g. with '--attach' or with
'-i/--stdin'.
```

Usage:

```
  kubectl run NAME --image=image [--env="key=value"] [--port=port] [--dry-run=server|client]
[--overrides=inline-json] [--command] -- [COMMAND] [args...] [options]
```

Rollout

Manage the rollout of one or many resources.

Valid resource types include:

* deployments
* daemonsets
* statefulsets

Examples:

```
# Rollback to the previous deployment
kubectl rollout undo deployment/abc

# Check the rollout status of a daemonset
kubectl rollout status daemonset/foo

# Restart a deployment
kubectl rollout restart deployment/abc

# Restart deployments with the app=nginx label
kubectl rollout restart deployment --selector=app=nginx

# Roll back to daemonset revision 3
kubectl rollout undo daemonset/abc --to-revision=3
```

Available Commands:

history	View rollout history
pause	Mark the provided resource as paused
restart	Restart a resource
resume	Resume a paused resource
status	Show the status of the rollout
undo	Undo a previous rollout

Usage:

```
kubectl rollout SUBCOMMAND [options]
```

Expose

Expose a resource as a new Kubernetes service.

Examples:
```
  # Create a service for a replicated nginx, which serves on port 80 and connects to the containers on port 8000
  kubectl expose rc nginx --port=80 --target-port=8000

  # Create a second service based on the above service, exposing the container port 8443 as port 443 with the name "nginx-https"
  kubectl expose service nginx --port=443 --target-port=8443 --name=nginx-https

  # Create a service for a replicated streaming application on port 4100 balancing UDP traffic and named 'video-stream'.
  kubectl expose rc streamer --port=4100 --protocol=UDP --name=video-stream

  # Create a service for an nginx deployment, which serves on port 80 and connects to the containers on port 8000
  kubectl expose deployment nginx --port=80 --target-port=8000
```

Options:

--cluster-ip='':
ClusterIP to be assigned to the service. Leave empty to auto-allocate, or set to 'None' to create a headless service.

--external-ip='':
Additional external IP address (not managed by Kubernetes) to accept for the service. If this IP is routed to a node, the service can be accessed by this IP in addition to its generated service IP.

-l, --labels='':
Labels to apply to the service created by this call.

--load-balancer-ip='':
IP to assign to the LoadBalancer. If empty, an ephemeral IP will be created and used (cloud-provider specific).

--name='':
The name for the newly created object.

--port='':
The port that the service should serve on. Copied from the resource being exposed, if unspecified

--protocol='':
The network protocol for the service to be created. Default is 'TCP'.

--selector='':
A label selector to use for this service. Only equality-based selector requirements are supported. If empty (the default) infer the selector from the replication controller or replica set.)

--target-port='':
Name or number for the port on the container that the service should direct traffic to. Optional.

--type='':
Type for this service: ClusterIP, NodePort, LoadBalancer, or ExternalName. Default is 'ClusterIP'.

Usage:
```
  kubectl expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol=TCP|UDP|SCTP] [--target-port=number-or-name] [--name=name] [--external-ip=external-ip-of-service] [--type=type] [options]
```

Taint

Update the taints on one or more nodes.

Examples:
```
  # Update node 'foo' with a taint with key 'dedicated' and value 'special-user' and effect 'NoSchedule'
  # If a taint with that key and effect already exists, its value is replaced as specified
  kubectl taint nodes foo dedicated=special-user:NoSchedule

  # Remove from node 'foo' the taint with key 'dedicated' and effect 'NoSchedule' if one exists
  kubectl taint nodes foo dedicated:NoSchedule-

  # Remove from node 'foo' all the taints with key 'dedicated'
  kubectl taint nodes foo dedicated-

  # Add a taint with key 'dedicated' on nodes having label mylabel=X
  kubectl taint node -l myLabel=X  dedicated=foo:PreferNoSchedule

  # Add to node 'foo' a taint with key 'bar' and no value
  kubectl taint nodes foo bar:NoSchedule
```

Options:

--all=false:
Select all nodes in the cluster

--allow-missing-template-keys=true:
If true, ignore any errors in templates when a field or map key is missing in the template. Only applies to golang and jsonpath output formats.

--dry-run='none':
Must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

--field-manager='kubectl-taint':
Name of the manager used to track field ownership.

-o, --output='':
Output format. One of: (json, yaml, name, go-template, go-template-file, template, templatefile, jsonpath, jsonpath-as-json, jsonpath-file).

--overwrite=false:
If true, allow taints to be overwritten, otherwise reject taint updates that overwrite existing taints.

-l, --selector='':
Selector (label query) to filter on, supports '=', '==', and '!='.(e.g. -l key1=value1,key2=value2). Matching objects must satisfy all of the specified label constraints.

Usage:
```
  kubectl taint NODE NAME KEY_1=VAL_1:TAINT_EFFECT_1 ... KEY_N=VAL_N:TAINT_EFFECT_N [options]
```

VIM - shortcut

Podmiana jednego wystąpienia słowa	Wciskamy / wpisujemy wyszukiwane słowo np /nginx, następnie wciskamy **cgn** i słowo na jakie chcemy zamienić. Aby przejść do kolejnego wystąpienia słów, przechodzimy do normal mode klawiszem **ESC**, naciskamy **n,** jeżeli dane wystąpienie również chcemy zamienić wciskamy kropkę (**.**)
Podmiana wszystkich wystąpień w dokumencie	:s/<search_term>/<replace_term>/option option: c - potwierdzenie zamiany g - zamień wszystkie wystąpienia i - ignoruje wielkość liter
Podmiana wszystkich wystąpień w obecnej linii	:s/<search_term>/</replace_term/option option: c - potwierdzenie zamiany g - zamień wszystkie wystąpienia i - ignoruje wielkość liter
Podmiana jednego/wszystkich wystąpień całego słowa	%s/\<first_word\>/replace_word/ Z %s podmienia wszystkie, s podmienia wystąpienie w linii gdzie jest kursor
Usunięcie linii do końca od kursora	D
Kopiowanie słowa pod kursorem	yy
Wklej poniżej/powyżej kursora	p lub P
Idź na początek/koniec linii	Początek: Shift + _ / Koniec Shift + $
Utwórz nową linię poniżej/powyżej kursora	Poniżej: o / Powyżej: Shift + O
Usuwanie określonej ilości linii	5dd - usuwa 5 kolejnych linii wliczając linię z kursorem
Usunięcie wszystkiego od kursora do końca pliku	dG
Skocz na początek/koniec słowa	początek b, koniec e